PG6 Group

SCHEDULE OPTIMIZATION PROJECT

BAO Project Assignment

WEN LAY, DANIEL (120 HOURS)
CHENAOUI, ISMAIL (120 HOURS)

Abstract

This project aims to generate a course timetable from a custom-created dataset using two methods: Genetic Algorithm (GA) and Ant Colony Optimization (ACO).

We have implemented three hard constraints and one soft constraint, focusing on a mono-objective optimization approach. The project leverages the Genetic Algorithm and ACO for timetable optimization.

The implementation involves a predefined dataset containing students, instructors, rooms, and timeslots that must meet specific constraints. For the Genetic Algorithm, we utilized the inspyred library with uniform crossover, tournament selection, and bit-flip mutation. For the Ant Colony Optimization, we used custom implementations focusing on pheromone updates and heuristic information.

Once the codes for both methods are developed, we proceed to optimize their hyperparameters using Bayesian Optimization with the Optuna library. This optimization is performed over 50 trials. After obtaining the best hyperparameter configuration, we conduct 31 iterations, each with a different seed, to ensure robustness.

The project aims to analyze the impact of hard and soft constraints on the solution quality and the relationship between various hyperparameters. The Wilcoxon signed-rank test is used to evaluate the consistency of the results across the 31 iterations for both GA and ACO.

This work provides insights into how constraints and hyperparameters affect the scheduling process, aiming for efficient resource allocation and conflict minimization in university timetabling.

Problem

Background

University course timetabling is a complex and critical task for educational institutions. It involves assigning courses to specific timeslots and rooms while considering a myriad of constraints such as instructor availability, student schedules, and room capacities. Efficient timetabling ensures optimal use of resources, minimizes conflicts, and enhances the educational experience for students and instructors.

The significance of this problem lies in its impact on the operational efficiency and overall satisfaction within educational institutions. Poorly designed timetables can lead to overcrowded classes, scheduling conflicts, and underutilized resources, which can hinder the learning process and cause frustration among students and faculty.

Research in this area has explored various optimization techniques to tackle the inherent complexity of the problem. Genetic Algorithms (GAs) and Ant Colony Optimization (ACO) are two prominent metaheuristic methods widely used for such combinatorial optimization problems. These methods are favored for their ability to provide high-quality solutions within reasonable computational times.

For instance, the work by Carter and Laporte (1998) explores various timetabling algorithms and highlights the complexity and NP-hard nature of the problem, which makes exact methods infeasible for large instances. More recent studies have applied metaheuristics such as GAs and ACO to find near-optimal solutions, demonstrating their effectiveness in handling the multifaceted constraints of university timetabling.

In this project, we aim to optimize a dataset that we created specifically for this study. The dataset includes detailed information on courses, rooms, timeslots, students, and instructors. The problem is approached as a mono-objective optimization problem, focusing on minimizing the number of constraint violations.

The scheduling optimization problem in this project involve the creation of an optimal schedule considering 3 hard constraints and 1 soft constraint. For that, we are going to explain with more details the definition of our problem, providing and including what we applicate.

DATASET.

The dataset was meticulously designed to reflect the real-world complexities of university timetabling. It includes various courses with specific requirements, available rooms with different capacities, timeslots, and detailed information about students and instructors.

First, we had to create our own dataset so we could prove the proper functioning for both methodologies used to afford the problem. We choose to afford the problem by this way due to that we were not able to find a dataset already created that fits with the nature of our problem. By this way, we instanced 5 tables with their respective attributes/features:

- Courses (200 courses): course_id, number_of_students, number_of_hours_per_week, instructor_id.
- Timeslots (25 timeslots): timeslot_id, day,start-time,end_time.
- Rooms (30 rooms): room_id,capacity.
- Instructors (80 instructors): instructor_id, courses, preferred_timeslots.
- Students (500 students): student_id, courses.

We also had to consider some restrictions while creating this dataset to make it more consistent and with common sense:

- Timeslots: are distributed in two hours from 9:00AM to 7:00PM, from Monday to Friday.
- Capacity in table Rooms: To avoid problems of rooms capacities we defined all rooms with a capacity of30.
- Course_id in table Courses: To avoid problems of courses with their respective rooms assigned we defined a minimum of 15 and a maximum of 30 students relied to a course.
- Number_of_hours_per week in table Courses: we defined that a course could have 2 or 4 hours as a duration.
- Foreign keys/attributes in Tables: knowing that the dataset was created randomly, we had to consider that the foreign keys instanced in the elements of the tables must match so we could ensure consistency to our dataset (for example, if an instructor has a course assigned it is obvious that it must match with instructor_id for that course instanced in table Courses).
- Courses in table Students: we defined a possible number between 3 and 6 that any student can have as courses.
- Preferred_timeslots in table Instructors: we defined a possible number between 2 and 4 that an instructor can have as preferred timeslots.

- courses in table Instructors: we defined a maximum number of 4 courses that an instructor can impart.

Here is an example of how our tables of our dataset are defined:

Courses Table.

Course ID	Number of Students	Number of Hours per Week	Instructor ID
C001	17	4	I001
C002	29	2	I002
C003	20	4	I003
C004	25	2	I004
C005	30	4	I005
C006	28	4	I001
C007	21	2	I002
C008	30	4	I003

Rooms Table.

Room ID	Capacity
R001	30
R002	30
R003	30
R004	30
R005	30
R006	30
R007	30
R008	30
R009	30
R010	30

Timeslots Table.

Timeslot ID	Day	Start Time	End Time
TS1	Monday	9:00 AM	11:00 AM
TS2	Monday	11:00 AM	1:00 PM
TS3	Monday	1:00 PM	3:00 PM
TS4	Monday	3:00 PM	5:00 PM
TS5	Monday	5:00 PM	7:00 PM
TS6	Tuesday	9:00 AM	11:00 AM
TS7	Tuesday	11:00 AM	1:00 PM
TS8	Tuesday	1:00 PM	3:00 PM

Instructors Table.

Instructor ID	Courses	Preferred Timeslots
I001	[C001, C006, C017]	[TS1, TS2, TS3]
I002	[C002, C007, C013, C018]	[TS4, TS5]
I003	[C003, C008, C014, C019]	[TS6, TS7, TS8]
I004	[C004, C009, C016, C020]	[TS9, TS10]
I005	[C005, C010, C015]	[TS11, TS12, TS13]
I006	[C011, C012]	[TS14, TS15]
...	...	...
I080	[C199, C200]	[TS24, TS25]

Students table.

Student ID	CoursesST
S001	[C001, C002, C003, C004, C005]
S002	[C001, C003, C005, C006, C007]
S003	[C002, C004, C006, C008, C009]
S004	[C003, C005, C007, C010, C011]
S005	[C004, C006, C008, C012, C013]
...	...
S500	[C198, C199, C200]

CONSTRAINTS HANDLED AND FITNESS FUNCTION.

As previously said, the objective is to generate a timetable that assigns to each course a timeslot and room focusing on minimizing the violations of some constraints that we defined, handling separately each type of constraint and particularly prioritizing the hard constraints to ensure a feasible timetable and using a soft constraint so we could difference between two feasible solutions that have the same amount of hard constraints violation. By this way, our problem will have a mono-objective characteristic.

Problem Definition

In this project, we aim to solve the university course timetabling problem using a custom dataset. The objective is to generate a timetable that assigns each course to a specific timeslot and room while minimizing the number of constraint violations. The constraints considered in this study are categorized into hard and soft constraints:

1. **Hard Constraints**: These must be strictly satisfied to ensure a feasible timetable.
    - **No Overlapping Classes in the Same Room(H_1)**: A room cannot be assigned more than one course at the same time.
    - **Instructor Assignment Conflicts(H_2)**: An instructor cannot be scheduled to teach more than one course at the same time.
    - **Student Schedule Conflicts(H_3)**: Students should not have overlapping classes.
2. **Soft Constraint**: These are desirable but not mandatory, providing a measure for optimizing beyond feasibility.
    - **Instructor Preferred Timeslots(S_1)**: Courses should be scheduled during the timeslots preferred by the instructors as much as possible.

Constraints and Objectives

- **Objective Function**: The primary goal is to minimize the total penalty incurred from constraint violations. The fitness function used in evaluating solutions is defined as:

The methodology applied to handle the previous constraints was using the penalty function method, increasing the penalty in one or other way if a non-compliance occurs.

Constraint Type	Description	Penalty Value
Hard Constraints		
Room-Time Conflict $(\boldsymbol{H_1})$:	A room cannot be assigned more than one course at the same time.	1.0
Instructor Conflict $(\boldsymbol{H_2})$:	An instructor cannot be scheduled to teach more than one course at the same time.	1.0
Student Conflict$(\boldsymbol{H_3})$:	Students should not have overlapping classes.	1.0
Soft Constraints		
Instructor Preference$(\boldsymbol{s_1})$:	Courses should be scheduled during the timeslots preferred by the instructors whenever possible.	0.5

Considering the previous constraints, the fitness function defined to be used in searching the best solution is:

The fitness function $f(\textbf{cand})$ can be expressed as:

$$f(\textbf{cand}) = \frac{H1+H2+H3+SP}{|C|\times 3}$$

Where:

- $H1 = \sum_{\text{course}\in C} \sum_{(\text{room},\text{time})} \mathbb{I}\{(\text{room}, \text{time}) \text{ is occupied by more than one course}\}$
- $H2 = \sum_{\text{course}\in C} \sum_{(\text{instructor},\text{time})} \mathbb{I}\{(\text{instructor}, \text{time}) \text{ is assigned to more than one course}\}$
- $H3 = \sum_{\text{course}\in C} \sum_{(\text{student},\text{time})} \mathbb{I}\{(\text{student}, \text{time}) \text{ has overlapping courses}\}$
- $SP = \sum_{\text{course}\in C} \sum_{\text{time}} \mathbb{I}\{\text{time} \notin \text{preferred times of assigned instructor}\} \times 0.5$

Here, $\mathbb{I}\{\cdot\}$ is an indicator function that returns 1 if the condition is true, and 0 otherwise.

Explanation:

- **Numerator:** $H1 + H2 + H3 + SP$ sums up all the conflicts and penalties for a given schedule.
- **Denominator:** $|C| \times 3$ normalizes this sum based on the total number of courses and the three types of conflicts (thus making the fitness value between 0 and 1, where lower values are better).

This formula is applied to each candidate schedule to determine its fitness based on how well it meets the hard and soft constraints.

Methodology

To solve this problem, we employed two metaheuristic algorithms: Genetic Algorithm (GA) and Ant Colony Optimization (ACO).

Genetic Algorithm (GA)

GA uses a population of candidate solutions, evolving them over generations through selection, crossover, and mutation to find an optimal or near-optimal solution.

Ant Colony Optimization (ACO)

ACO simulates the foraging behavior of ants to find optimal paths, using pheromone trails to guide the search process towards promising solutions.

Algorithm design

As we said in previous sections, the metaheuristics selected for solving the main problem are GA and ACO. We have chosen these two methods due to their adaptability, robustness, and the huge capacity that both have for searching in large complex solution spaces.

However, knowing that they have differences we wanted to prove that we can achieve our objectives by using these two methods so we could better understand their ways of optimization and compare between both to know which one fits or suits better to our main problem.

We can also observe that they also have some commonalities due to the main objective that share, so it is logical that they share some points as it is the same fitness function: even if it is implemented in different ways for each method both ways have the same parameters configured and return the same supposed value.

Regarding the codification used for both methods, knowing that they are two different ways for solving our problem so we had to implement different methods for each of them to make sure that we could achieve the best solution by far.

We need to consider that for both algorithms there are implemented functions that allows the visualization of information related to each algorithm after execution. This help and guide us to have relevant information about some parameters and serve as a guideline for understanding better the obtained results and for future parameters modifications if necessary.

Following this, we shall explain more detailly each metaheuristic.

GENETIC ALGORITHM.

Given the procedure for the development of the genetic algorithm, we divided the implemented algorithm into the following sections:

- **Candidates generated**: using the function generate_candidates, we obtain/generate as candidate's tuples that contain information about a course assignment. The format of each tuple will be course_id, room_id, timeslot_id, instructor_id, and student_list.

```
Function generate_candidate:
    Initialize candidate
    For each course in courses:
        Select room (random)
        Select timeslot (random)
        List students in course
        Add (course_id, room_id, timeslot_id, instructor_id, students_list) to candidate
    Return candidate
```

- **Solutions generated**: a list of tuples that represents information about a specific course with the belonged information. For more information it is possible to print for each room the courses assigned to understand better the solution.

- **Fitness function**: it is defined by the constraints mentioned in the previous section. For this, we developed the fitness function with the constraints and other functions to check constraints violations and complete the method.

```
Function Fitness:
    Evaluate constraints:
        Evaluate H1
        Evaluate H2
        Evaluate H3
        Evaluate S1 (soft penalty)
    Hard conflicts = sum(Evaluation of H's)
    Normalize fitness = (Hard conflicts + Soft penalty) / max_possible_conflicts
    Return fitness values
```

  We also defined functions that evaluate and calculate constraints compliance to a future use for plotting data and take information about constraints evaluation.

- **Stopping criteria method**: instead of using a stopping criteria (terminator) proposed by inspired we developed our own customized stopping criteria. We designed it in such a way that stops the algorithm if after 10 consecutive generations there is not a significant variation in the results generated.

- **Customized observer**: as we did for the stopping criteria, we implemented our own observer so we could also gather information instead of just printing the belonged fitness values (worst, best, median and average) for each iteration/generation.
- **Parameters information**: knowing that we did not have too much time for developing both algorithms we preferred to select just specific parameters of the genetic algorithm so we could face the problem. Those parameters used are the following:

```
GA Parameters Used:
    Selector = Tournament Selector
    Variator = Uniform crossover and Bit Flip Mutation
    Replacer = Generational replacer
    Terminator = custom terminator
    Observer = custom observer
```

ANT COLONY OPTIMIZATION.

First, we started by setting several parameters that we are going to use for the development. So, we instanced the number of ants, the number of iterations, alpha and beta and the evaporation rate.

Pheromones levels are initialized uniformly for each combination of courses, rooms and timeslots (to ensure that all possible assignments have an equal probability at the start). The heuristic information is also initialized uniformly.

The algorithm computes the probability of assigning courses to rooms and timeslots using both pheromone levels and heuristic information (balancing in that way between exploration and exploitation). Based on the computed probabilities a solution is built randomly selecting rooms and timeslots for each course (simulating in that way path's finding behavior of ants).

After constructing solutions, we proceed to update the pheromones and to calculate the fitness by analyzing constraints compliance.

The algorithm runs for a specific number of iterations where in each one of them do the following steps: compute probabilities, generate solutions, evaluate fitness, update pheromones and collect statistics. At the end of these iterations the best solution is identified and a detailed report of the schedule and the algorithm over iterations are visualized.

In the following image we can see a better understanding of our algorithm.

<u>Pseudocode of the algorithm.</u>

```
ACO Pseudocode:
    Initialize pheromones and calculate heuristic
    Compute probabilities (based on pheromones and heuristic)
    Build solution (based on probabilities)
    Update pheromones
    Calculate constraints compliance:
        Calculate hard constraints compliance:
            Overlapping classes in the same room (H1)
            Instructors assigned to more than one class (H2)
            Students having overlapping classes (H3)
        Calculate instructor's preferences compliance (S1)

Algorithm operation:
    For n iterations:
        Compute probabilities
        Generate and evaluate solution
        Update pheromones based on the fitness values
        Collect statistics to track algorithm's performance over generations
```

Implementation Details

The algorithms were implemented using Python, leveraging libraries such as numpy for numerical operations and pandas for data manipulation. Specific tools and libraries used include:

- **Genetic Algorithm**:
  - **Library:** inspired,Optuna
  - **Coding Framework:**

    ```
    import inspyred
    from inspyred import ec
    import random
    from time import time
    import matplotlib.pyplot as plt
    import pandas as pd
    ```

- **Ant Colony Optimization**:
  - **Library:** Custom implementation, Optuna
  - **Coding Framework:**

    ```
    import numpy as np
    import random
    import matplotlib.pyplot as plt
    import pandas as pd
    ```

Hyperparameter Optimization with Optuna

Optuna is an automatic hyperparameter optimization software framework, particularly designed for machine learning. It was employed in this project to optimize the hyperparameters of both GA and ACO, ensuring the best possible performance of these algorithms for the given problem.

Genetic Algorithm (GA)

Hyperparameters Tuned:

- Population Size (pop_size)
- Maximum Generations (max_generations)
- Mutation Rate (mutation_rate)
- Crossover Rate (crossover_rate)

Implementation in Optuna:

1. **Objective Function:** Define an objective function that runs the GA with a given set of hyperparameters and returns the best fitness value.
2. **Search Space:** Define the search space for the hyperparameters using Optuna's distribution classes.
3. **Optimization:** Use Optuna's study to optimize the objective function over multiple trials, leveraging the Tree-structured Parzen Estimator (TPE) sampler for efficient exploration.

Ant Colony Optimization (ACO)

Hyperparameters Tuned:

- Number of Ants (num_ants)
- Number of Iterations (num_iterations)
- Alpha (alpha)
- Beta (beta)
- Evaporation Rate (evaporation_rate)

Implementation in Optuna:

1. **Objective Function:** Define an objective function that runs the ACO with a given set of hyperparameters and returns the best fitness value.
2. **Search Space:** Define the search space for the hyperparameters using Optuna's distribution classes.
3. **Optimization:** Use Optuna's study to optimize the objective function over multiple trials, leveraging the TPE sampler for efficient exploration.

Experiments Design

Experiments Design for Genetic Algorithm Hyperparameter Tuning

Parameter Settings

In the context of optimizing a Genetic Algorithm (GA) for course scheduling, several key hyperparameters significantly impact the algorithm's performance and effectiveness. These parameters include:

1. **Population Size (pop_size)**: This is the number of candidate solutions in each generation. A larger population size can enhance the diversity of solutions and improve the exploration of the search space. However, it also increases computational cost and memory usage.
2. **Maximum Generations (max_generations)**: This parameter sets the maximum number of iterations the algorithm will execute. It provides a stopping criterion to ensure the algorithm does not run indefinitely. More generations can lead to better solutions but also increase computation time.
3. **Mutation Rate (mutation_rate)**: This rate determines how frequently random mutations are applied to individuals in the population. A higher mutation rate can introduce more genetic diversity and help escape local optima, but it may also disrupt good solutions if too high.
4. **Crossover Rate (crossover_rate)**: This specifies the proportion of the population that will undergo crossover. Crossover combines two parent solutions to produce offspring, promoting the exchange of genetic material and potentially generating better solutions. A high crossover rate facilitates exploration, while a lower rate emphasizes exploitation.
5. **Tournament Selection Size**: This parameter is used during the selection process to choose the best individuals from the population for crossover.

The tournament size in the tournament selection method was fixed at 5 in our genetic algorithm. This was not an error, but rather a design decision in the code. By keeping certain parameters fixed, we can simplify the optimization process. If too many parameters are optimized simultaneously, it might lead to instability or increased computational complexity. By fixing the tournament size, we reduce the dimensionality of the hyperparameter search space, making the optimization problem easier to handle. The value of 5 have been chosen based on previous experiments that suggested it was a reasonable choice for this problem. Tournament selection with a size of 5 can provide a good balance between exploration (trying new solutions) and exploitation (refining known good solutions).

Optuna was used to optimize the most impactful hyperparameters: population size, number of generations, mutation rate, and crossover rate. These hyperparameters often have a more significant impact on the performance of the genetic algorithm compared to the tournament size. Thus, the decision was to focus the optimization effort on these key hyperparameters.

Experimental Setup

The experiments are designed to compare different configurations of the GA and evaluate their performance on the course scheduling problem. The setup includes the following steps:

1. **Hyperparameter Optimization with Optuna**: The Optuna library, utilizing Bayesian optimization, is used to find the best combination of hyperparameters. The goal is to minimize the fitness value, which represents the cost of the solution (lower is better).
2. **Evaluation Metrics**: The primary metric is the fitness value, which aggregates various constraints and preferences. Additionally, compliance with hard constraints (e.g., no overlapping classes, no instructor conflicts, no student conflicts) and soft constraints (e.g., instructor preferences) is tracked.
3. **Statistical Tests**: To ensure the robustness of the results, the Wilcoxon signed-rank test is employed to compare the fitness values across different seeds. This non-parametric test helps verify that the observed differences are statistically significant.
4. **Number of Runs**: The GA is executed 31 times with different random seeds to ensure that the results are not dependent on a specific initialization. This provides a more comprehensive evaluation of the algorithm's performance.
5. **Data Collection**: Results from each run, including best, worst, average, median, and standard deviation of fitness values, are collected for analysis.

Data Description

The data instances for the experiments include:

- **Courses**: Each course has a unique ID and is associated with an instructor.
- **Rooms**: Each room has a unique ID and a capacity.
- **Timeslots**: Each timeslot has a unique ID.
- **Students**: Each student is associated with a list of course IDs they are enrolled in.
- **Instructors**: Each instructor has a unique ID and a list of preferred timeslots.

These data instances are used to simulate the scheduling problem and evaluate the performance of the GA.

Results Summary

Table 1: Hyperparameters optimization for GA using Bayesian Optimization.

number	value	params_crossover_rate	params_max_generations	params_mutation_rate	params_pop_size
0	0.2791666666666667	0.7993292420985183	193	0.3686770314875885	106
1	0.26166666666666666	0.9330880728874675	73	0.03846097	73
2	0.21	0.9849549260809971	156	0.020086402204943198	140
3	0.25166666666666665	0.5917022549267169	82	0.09909423	175
4	0.3025	0.64561457	129	0.22165305913463673	95
5	0.30416666666666664	0.6831809216468459	71	0.1531508777822569	142
6	0.2841666666666667	0.7571172192068059	168	0.10784015325759626	118
7	0.27166666666666667	0.5852620618436457	57	0.3076969774317048	139
8	0.31666666666666665	0.9041986740582306	193	0.4831596962065341	59
9	0.30583333333333335	0.7200762468698007	64	0.34527418299095686	95
10	0.175	0.9775458067418015	134	0.01369819	194
11	0.17833333333333334	0.9933080661316598	137	0.01723348	195
12	0.22083333333333333	0.8488955341397639	115	0.2004902577865462	196
13	0.17333333333333334	0.9847131373438699	122	0.025167150341825872	196
14	0.2575	0.8780863884590011	103	0.09675824	174
15	0.20666666666666667	0.9477419509093674	98	0.13611740342205592	169
16	0.25583333333333336	0.8264648750204893	149	0.06259391	155
17	0.18916666666666668	0.9314071361143548	115	0.1682372394457161	199
18	0.1775	0.9994280965387005	144	0.2535718428693177	181
19	0.29333333333333333	0.7764209924043369	165	0.4576652671108272	160
20	0.225	0.8705300875955625	93	0.08622437	182
21	0.3016666666666667	0.5138903266349962	141	0.26317620852704104	185
22	0.18333333333333332	0.9920819566183433	120	0.4219127352613727	187
23	0.1875	0.9546918630818175	132	0.28684647433856086	164
24	0.18166666666666667	0.8998247946580235	176	0.22505113287024936	199
25	0.21166666666666667	0.95840456	147	0.0467729	156
26	0.21166666666666667	0.9074043391479311	106	0.41380840614503284	181
27	0.205	0.9986947933052505	125	0.18390324014040937	148
28	0.2683333333333333	0.8318857020092956	150	0.13202118499463472	130
29	0.25333333333333335	0.80064409	183	0.36507093399736024	190
30	0.20083333333333334	0.9689416658665593	160	0.06222945	173
31	0.185	0.99593486	138	0.011417814418010683	189
32	0.18166666666666667	0.93625577	137	0.02970545	200
33	0.23583333333333334	0.9250740114025219	110	0.062462520896178174	178
34	0.19583333333333333	0.9861085034576407	126	0.01571024	191
35	0.22083333333333333	0.9676416747700493	88	0.07996605	163

36	0.21333333333333335	0.8805991604929908	141	0.04073338	193
37	0.18083333333333335	0.92295385	121	0.13142613724194024	180
38	0.21166666666666667	0.9717681437099783	157	0.11145168934588404	112
39	0.29833333333333334	0.6367861393932244	136	0.31497413071874114	98
40	0.2991666666666667	0.7072958782974688	129	0.037141176642638926	170
41	0.2683333333333333	0.9201785504472138	120	0.23649896837951687	82
42	0.20666666666666667	0.9436617154394085	145	0.11756730886940278	177
43	0.18416666666666667	0.9998312946381512	121	0.14893385468973686	184
44	0.2775	0.97319558	111	0.07073582	51
45	0.23166666666666666	0.8966936028659661	132	0.19951539523717576	194
46	0.21583333333333332	0.9565032573663413	151	0.04735093	150
47	0.21916666666666668	0.8640671881145091	98	0.010888210305591471	168
48	0.245	0.9268854187060476	77	0.10052016533906628	130
49	0.18833333333333332	0.9789819667095292	51	0.16561126252442077	178

After conducting the hyperparameter optimization for the Genetic Algorithm with Bayesian Optimization, the best configuration parameters were identified as:

- Population Size (pop_size): 196
- Maximum Generations (max_generations): 122
- Mutation Rate (mutation_rate): 0.025167150341825872
- Crossover Rate (crossover_rate): 0.9847131373438699
- Tournament Selection Size: 5 (Fix parameter)
- Best Trial: 13
- Best Fitness: 0.1733333333333

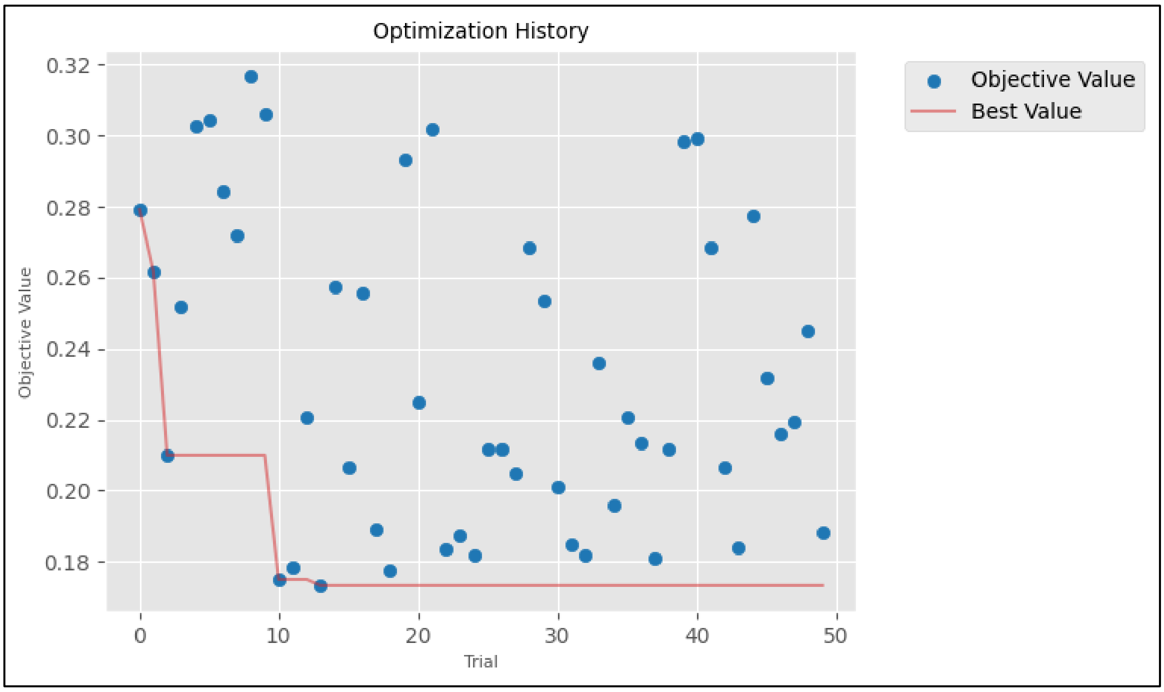

Figure 1. Optimization History of the Genetic Algorithms Hyperparameters.

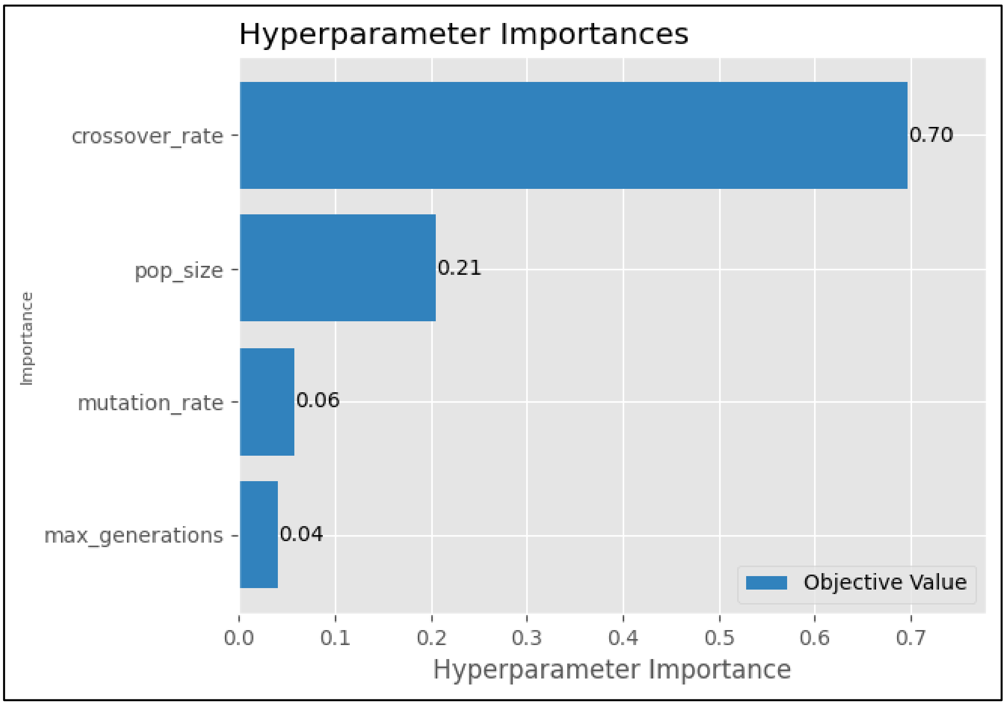

Figure 2. Hyperparameters importance by percentage.

In the Genetic Algorithm (GA), the importance of the hyperparameters was determined as follows:

- Crossover Rate: 70%
- Population Size: 21%
- Mutation Rate: 6%
- Maximum Generations: 4%

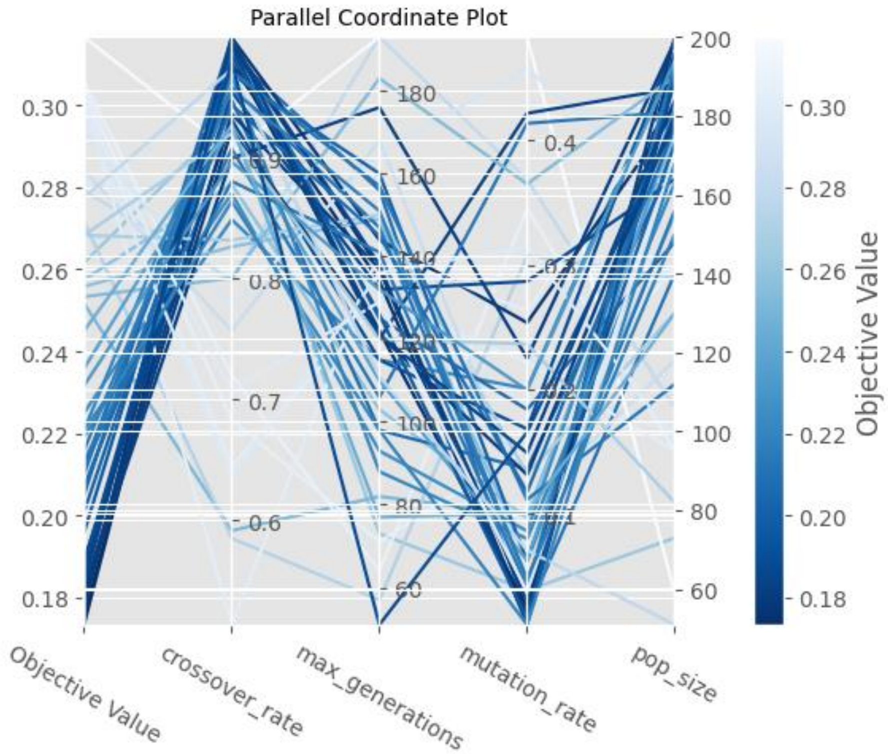

Figure 3. Parallel Coordinate Plot of the relationships between the four hyperparameters for the GA.

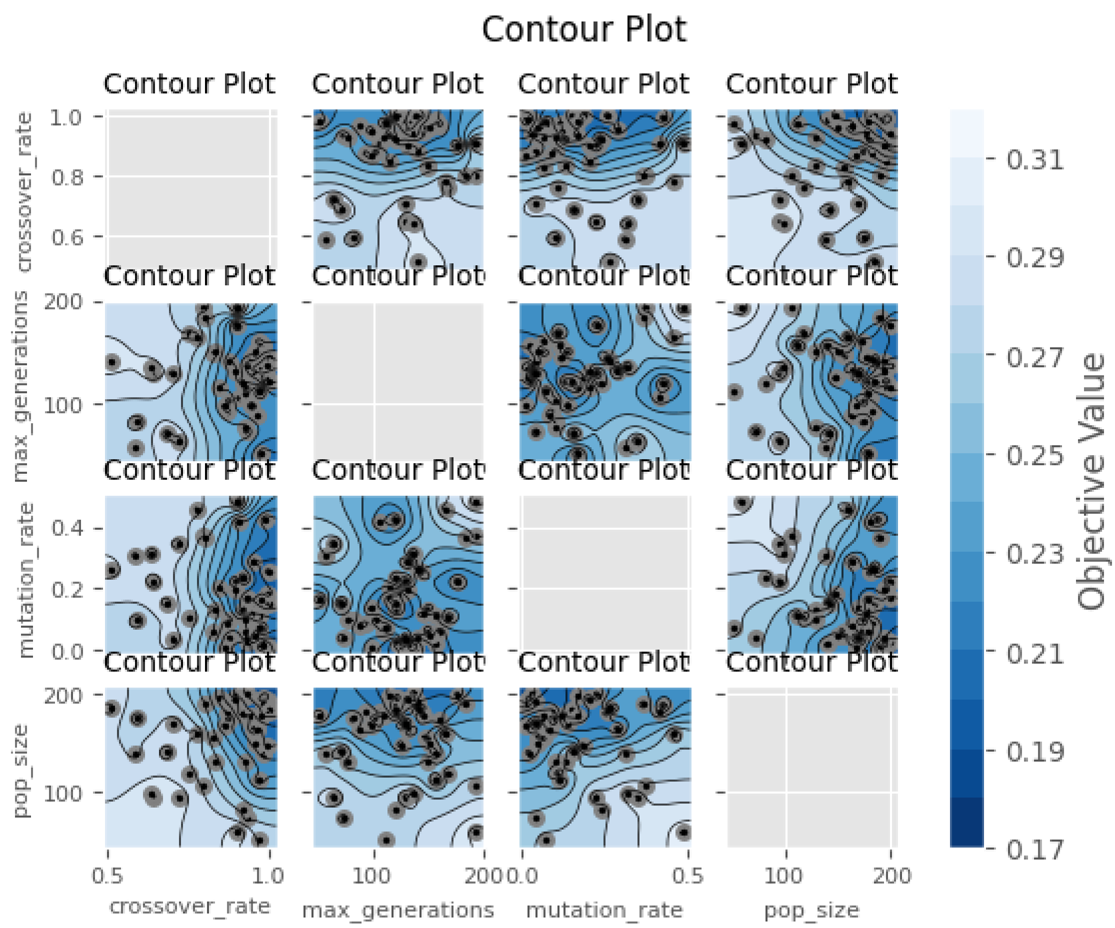

Figure 4. Contour plot for GA.

The contour plot provides a detailed view of how two hyperparameters interact with each other and their combined effect on the objective value. This helps in understanding the parameter space better.

<u>31 repetitions with the best configuration parameters found for GA.</u>

```
# Parameters obtained from hyperparameter tuning
best_params = {
    'pop_size': 196,
    'max_generations': 122,
    'mutation_rate': 0.025167150341825872,
    'crossover_rate': 0.9847131373438699
}
```

Wilcoxon signed-rank test p-value for GA: 9.313225746154785e-10

The extremely low p-value indicates that the genetic algorithm's best fitness values are consistent across different seeds, reinforcing the reliability of the hyperparameters tuned. This suggests that the tuned parameters result in stable performance of the algorithm across multiple runs.

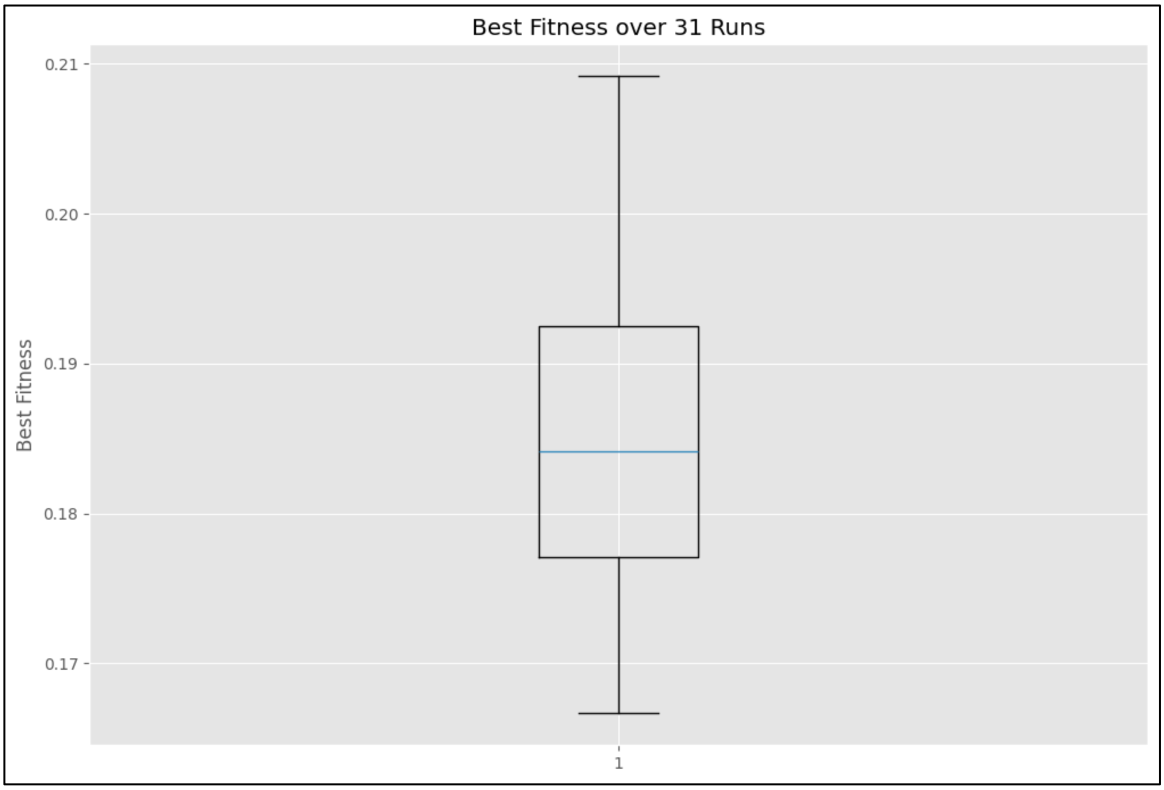

Figure 5. Best Fitness over 31 runs.

This figure illustrates the best fitness values achieved in the final generation of each of the 31 runs. Additionally, it supports the analysis performed using the Wilcoxon signed-rank test.

Using these parameters, the GA was run 31 times, and the results were as follows:

seed	best_fitness	worst_fitness	average_fitness	median_fitness	std_dev_fitness
1	0.20416667	0.20416667	0.20416667	0.20416667	0
2	0.17916667	0.17916667	0.17916667	0.17916667	0
3	0.18833333	0.18833333	0.18833333	0.18833333	0
4	0.17583333	0.17583333	0.17583333	0.17583333	0
5	0.18416667	0.18416667	0.18416667	0.18416667	0
6	0.185	0.185	0.185	0.185	0
7	0.19666667	0.19666667	0.19666667	0.19666667	0
8	0.17666667	0.17666667	0.17666667	0.17666667	0
9	0.16833333	0.16833333	0.16833333	0.16833333	0
10	0.18916667	0.18916667	0.18916667	0.18916667	0
11	0.1775	0.1775	0.1775	0.1775	0
12	0.20416667	0.20416667	0.20416667	0.20416667	0
13	0.175	0.175	0.175	0.175	0
14	0.16916667	0.16916667	0.16916667	0.16916667	0
15	**0.16666667**	**0.16666667**	**0.16666667**	**0.16666667**	**0**
16	0.19166667	0.19166667	0.19166667	0.19166667	0
17	0.19333333	0.19333333	0.19333333	0.19333333	0
18	0.18416667	0.18416667	0.18416667	0.18416667	0

19	0.18166667	0.18166667	0.18166667	0.18166667	0
20	0.16833333	0.16833333	0.16833333	0.16833333	0
21	0.1825	0.1825	0.1825	0.1825	0
22	0.18333333	0.18333333	0.18333333	0.18333333	0
23	0.195	0.195	0.195	0.195	0
24	0.1875	0.1875	0.1875	0.1875	0
25	0.20583333	0.20583333	0.20583333	0.20583333	0
26	0.18916667	0.18916667	0.18916667	0.18916667	0
27	0.175	0.175	0.175	0.175	0
28	0.20916667	0.20916667	0.20916667	0.20916667	0
29	0.18666667	0.18666667	0.18666667	0.18666667	0
30	0.18	0.18	0.18	0.18	0
31	0.2025	0.2025	0.2025	0.2025	0

As you can see from the table, the best performing one was with seed 15.

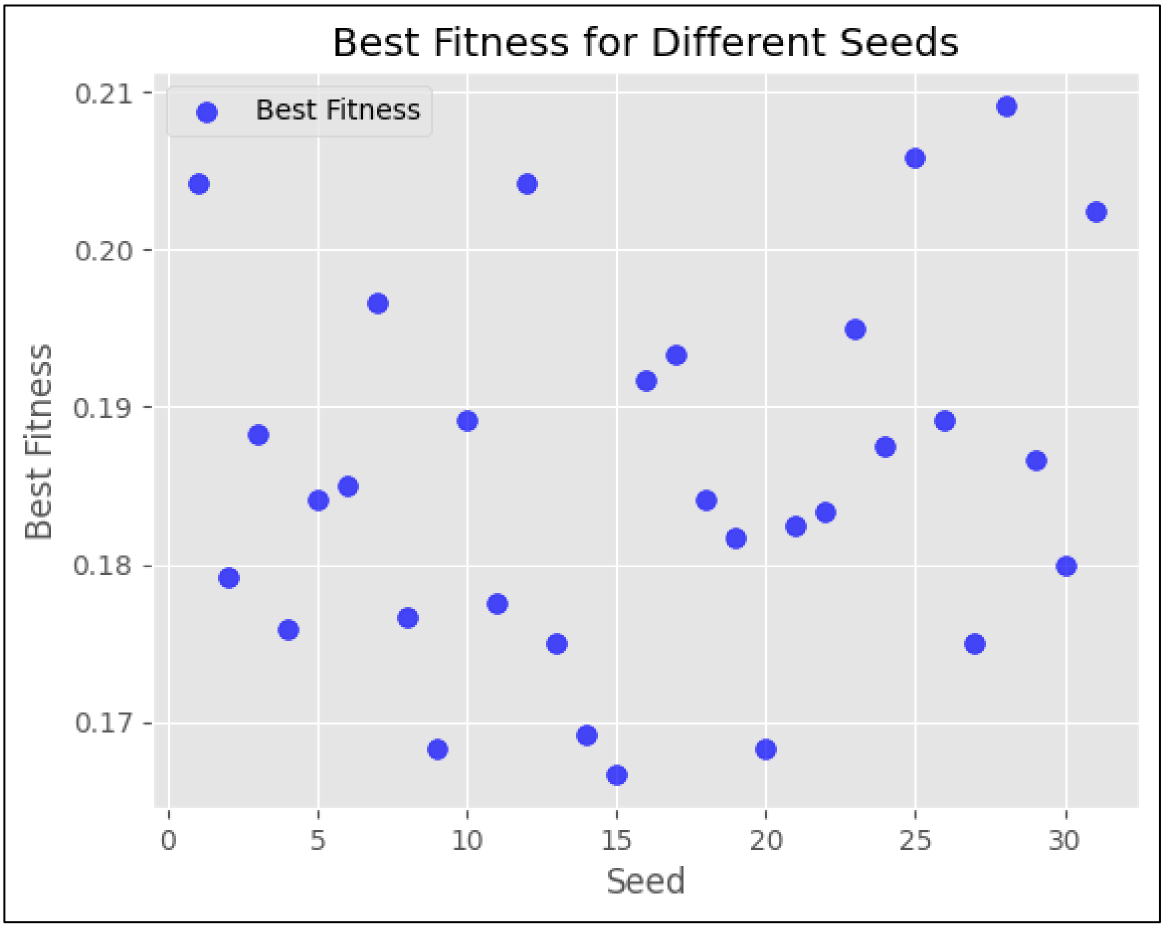

Figure 6. Best Fitness for Different Seeds for all 31 iterations for the GA.

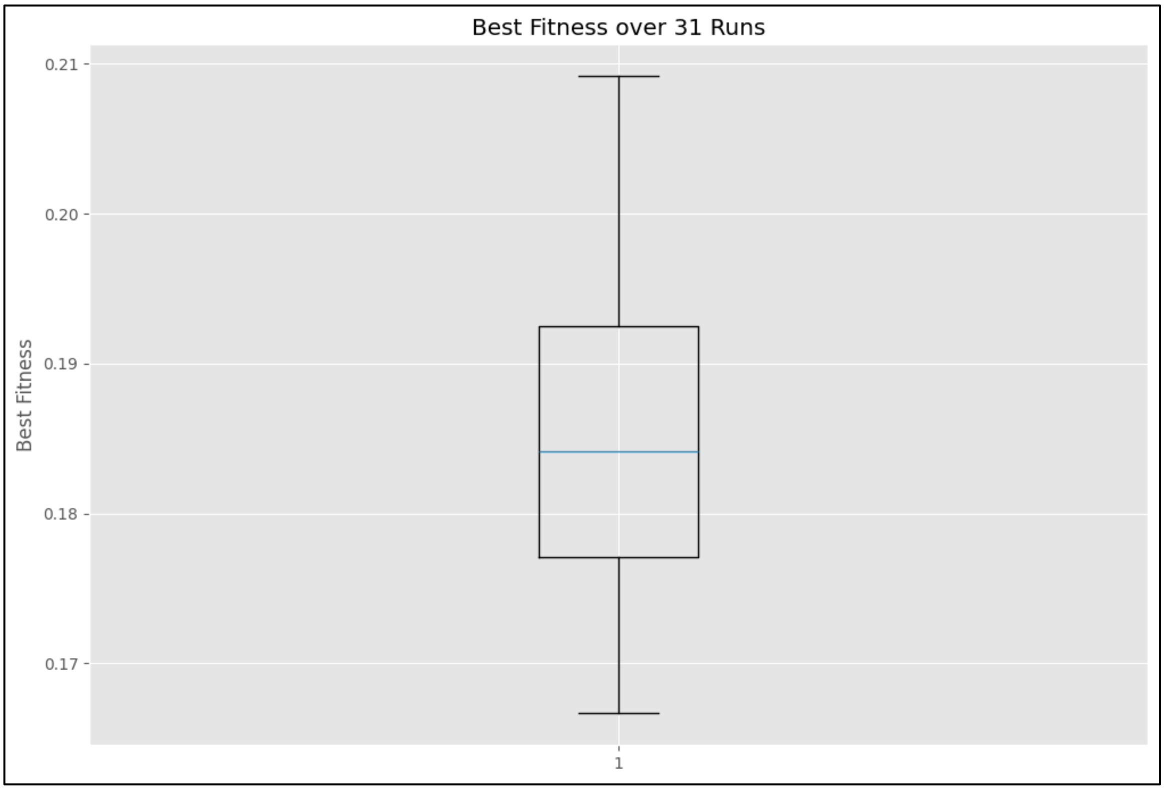

Figure 7. Best Fitness over 31 runs.

This figure illustrates the best fitness values achieved in the final generation of each of the 31 runs. Additionally, it supports the analysis performed using the Wilcoxon signed-rank test.

The GA produced schedules with high compliance to constraints:

- **Compliance with No Overlap Conflicts (Hard Constraint 1):** *97.50%*
- **Compliance with Instructor Conflicts (Hard Constraint 2):** *100.00%*
- **Compliance with Student Conflicts (Hard Constraint 3):** *92.50%*
- **Instructor Preference Compliance (Soft Constraint):** *45.00%*

The solution provided by the GA with the best configuration (and seed 15) achieved high compliance with all constraints and an optimal fitness value of **0.16333.**

These results demonstrate the effectiveness of the GA in generating feasible schedules that comply with the defined constraints.

The solution provided by the genetic algoritm with the best configuration compliances the H_1(97.50%), H_2(100%), H_3(92.50%) and the soft constraint (s1) by 45%. With a best fitness of 0.16333. (All these results were got with the seed 15)

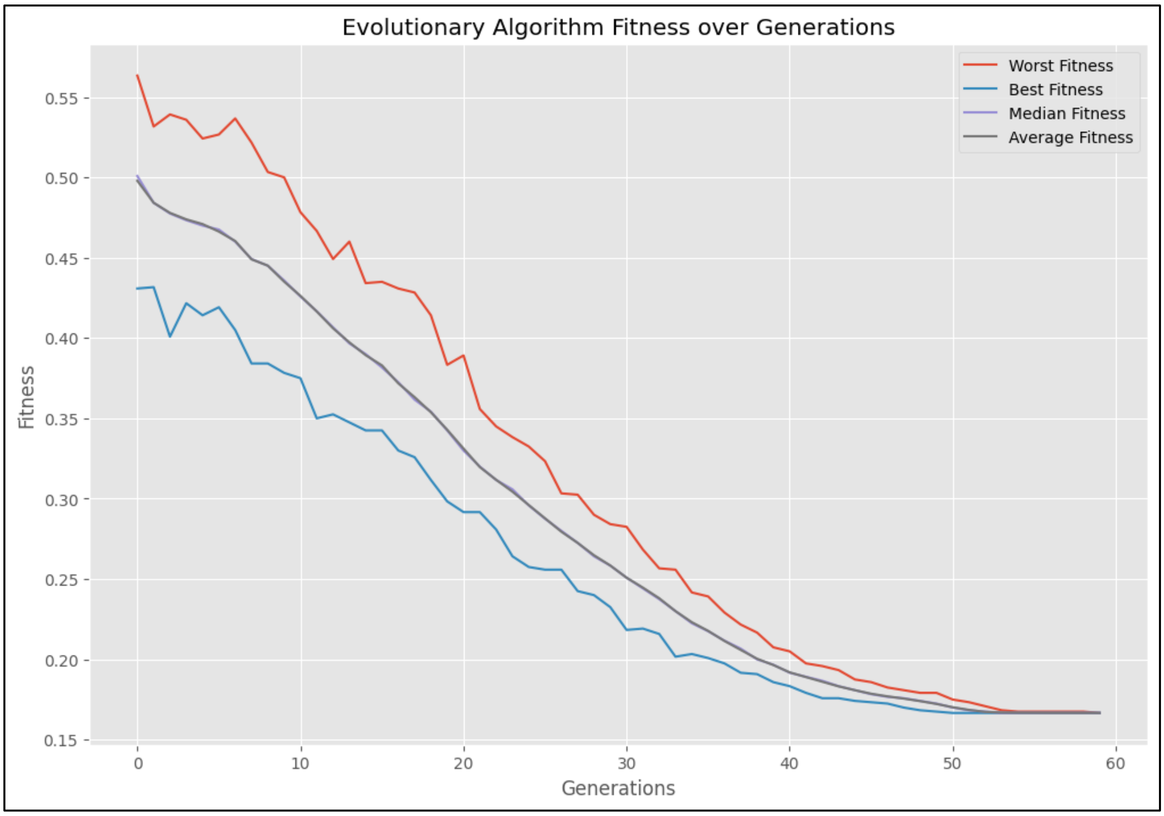

Figure 8. Evolutionary Algorithm Fitness over Generations. This figure shows the plot comparison of the best, median, average, and worst fitness values over generations. Note that the median and average values are about the same, making them barely noticeable.

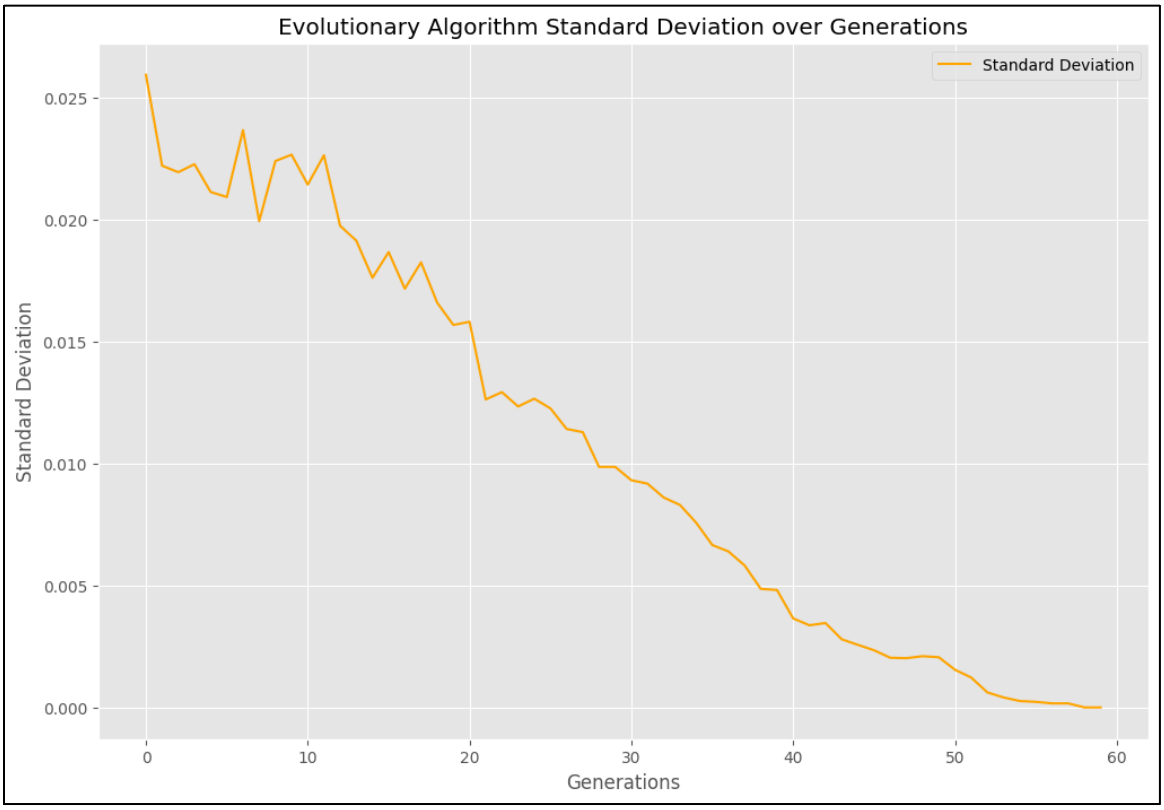

Figure 9. Evolutionary Algorithm Standard Deviation over Generations. This figure is useful to identify when the stopping criteria might stop the generations. After 10 generations of no significant improvement, the algorithm is expected to stop.

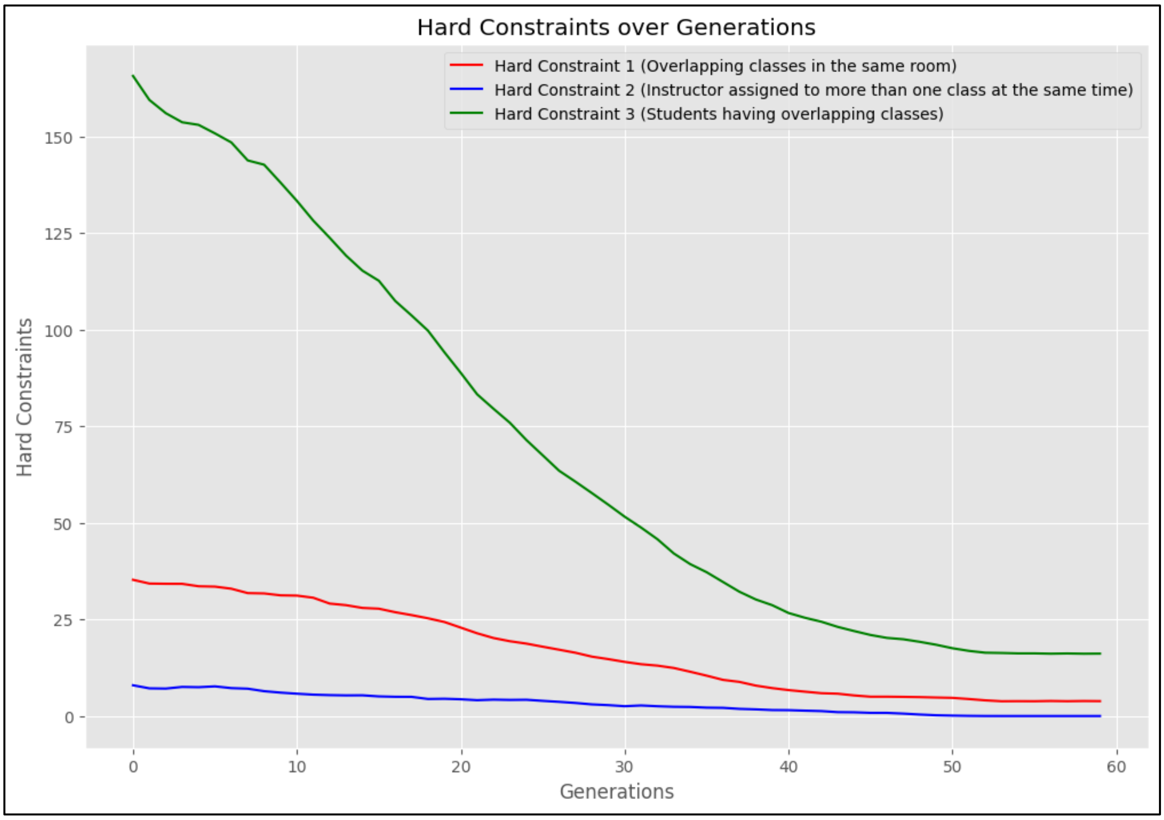

Figure 10. Hard Constraints over Generations.

This plots the evolution of the hard constraints over the generations and how it is improving over time. As you can see, the algorithm performs better for the Hard Constraint 1 and Hard Constraint 2. While the worst performance hard constraint is the number 3.

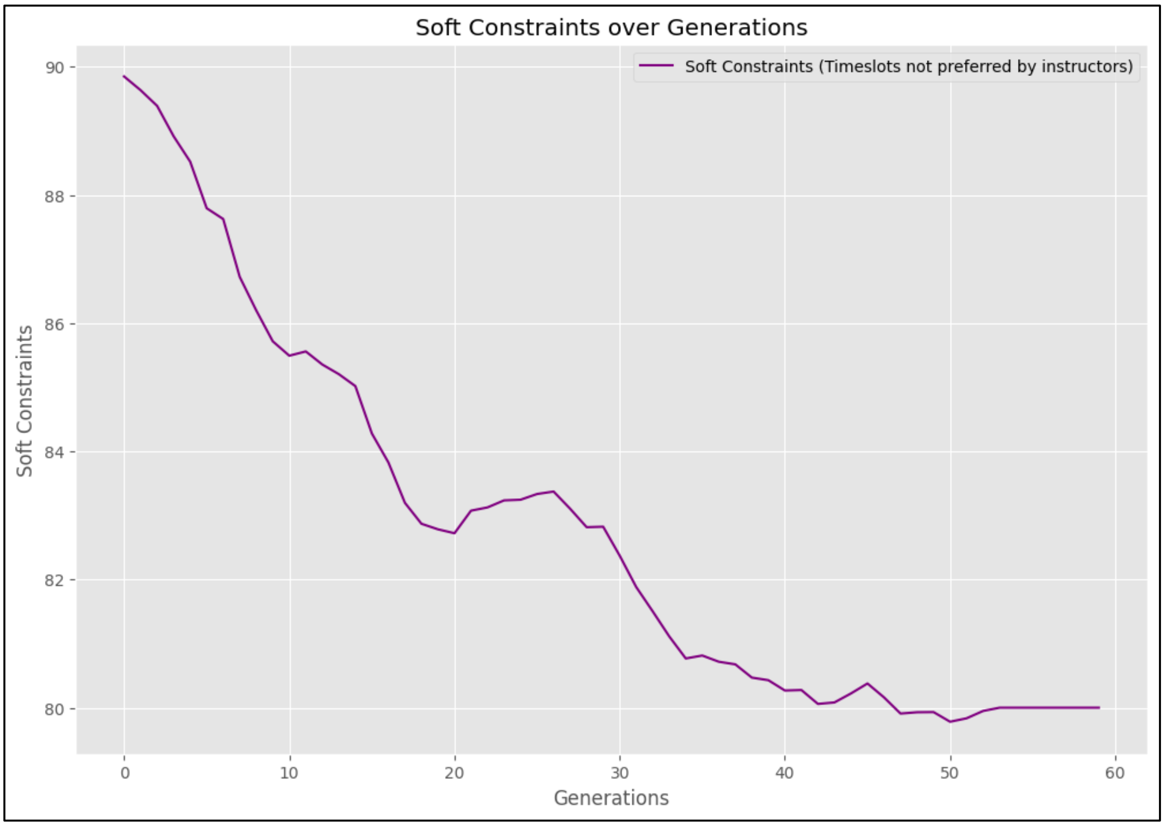

Figure 11. Soft Constraint over Generations. In this case we only have one soft constraint. ▫ This figure shows the compliance with the soft constraint (timeslots preferred by instructors) over time.

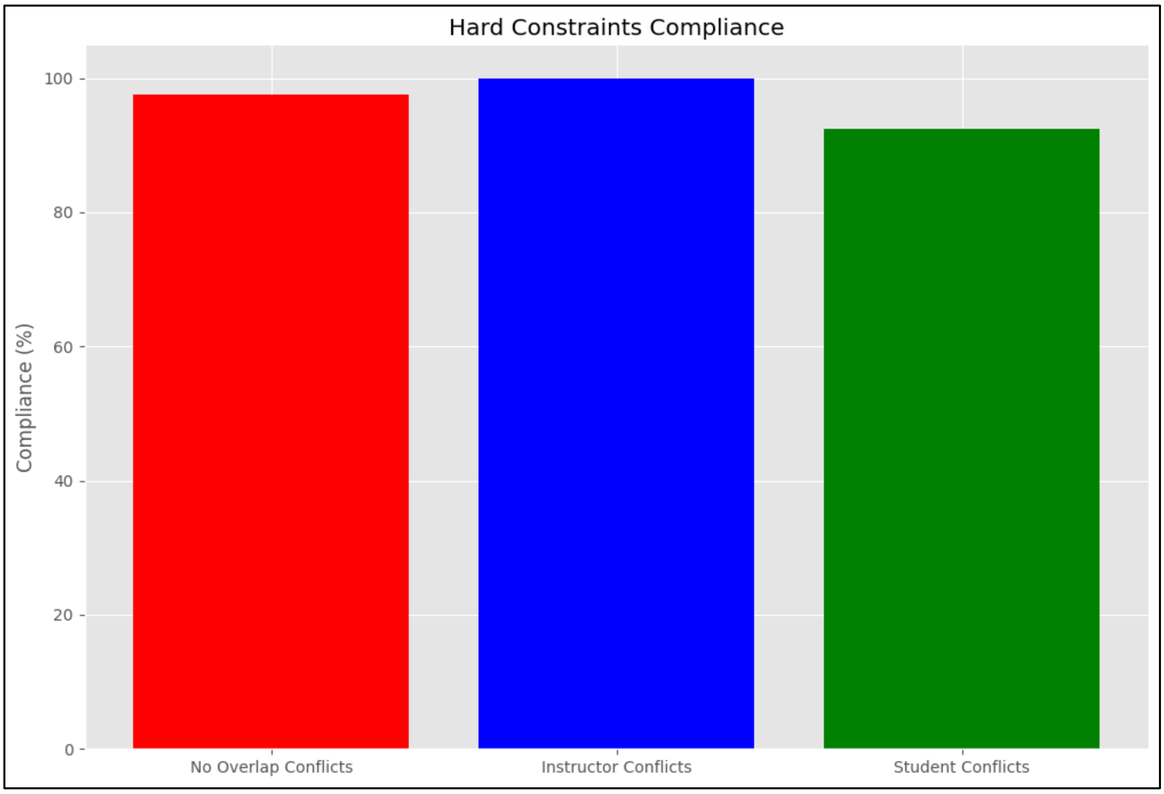

Figure 12. Hard Constraints Compliance for the best performing seed.

This shows how much percentage of the hard constraints criteria is met.
In this case:

Compliance with no overlap conflicts (Hard Constraint 1): 97.50%
Compliance with instructor conflicts (Hard Constraint 2): 100.00%
Compliance with student conflicts(Hard Constraint 3): 92.50%

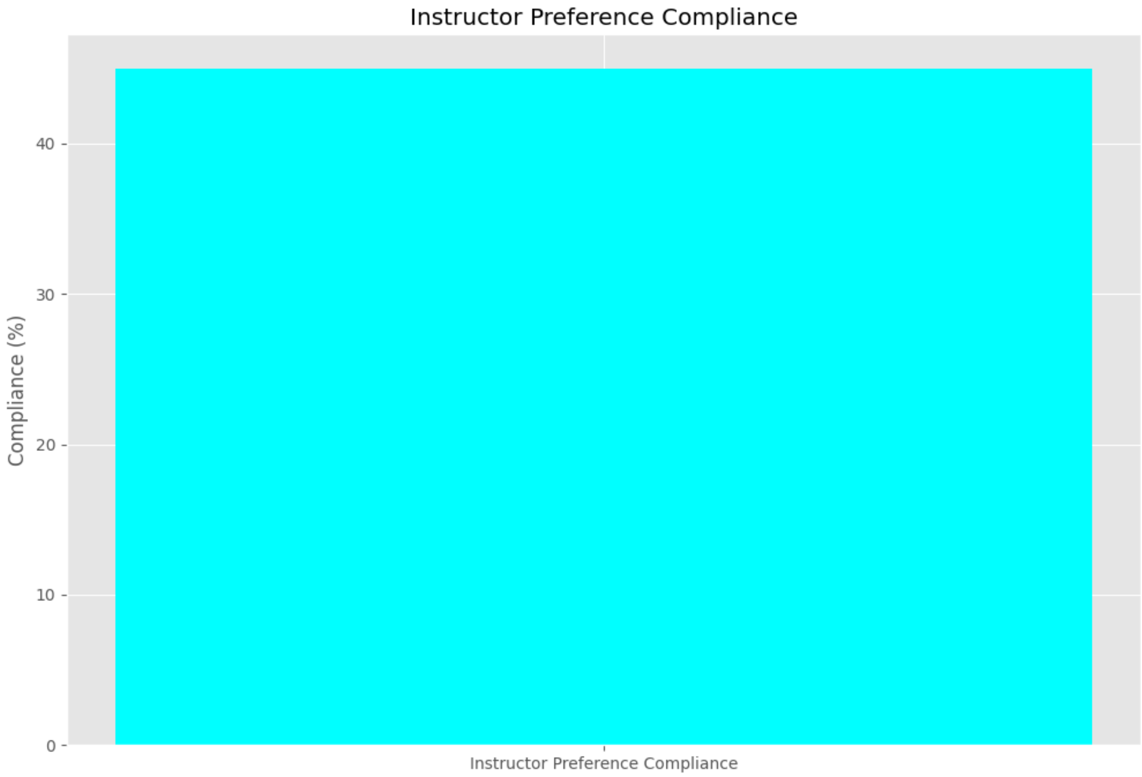

Figure 13. Instructor Preference Compliance in percentage %.

This figure shows the compliance with instructor preferences (Soft Constraint), which is 45.00%.

The hyperparameter tuning and subsequent analysis provided valuable insights into the performance and robustness of the Genetic Algorithm for course scheduling. The combination of Optuna for hyperparameter tuning and statistical tests for result validation ensures a reliable and efficient approach to solving complex scheduling problems.

Ant Colony Optimization Experimental Results

number	value	params_alpha	params_beta	params_evaporation_rate	params_num_ants	params_num_iterations
0	0.42916666666666664	2.3299848545285125	3.3946339367881464	0.22481491235394924	50	193
1	0.4475	2.665440364437338	3.404460046972835	0.6664580622368363	32	58
2	0.42	2.5811066020010545	1.8493564427131046	0.24545997376568052	21	196
3	0.43166666666666664	1.8118910790805947	2.727780074568463	0.3329833121584336	34	95
4	0.43166666666666664	1.2303616213380453	2.465447373174767	0.46485598737362877	69	71
5	0.43083333333333335	1.785586096034029	3.36965828	0.13716033017599819	83	80
6	0.4125	0.6626289824631988	4.7955421490133325	0.8725056264596475	69	75
7	0.4166666666666667	0.7441802850159597	3.7369321060486276	0.45212199499168104	85	95
8	0.42333333333333334	0.5859713	4.637281608315128	0.3070239852800135	29	124
9	0.42	1.800170052944527	3.1868411173731186	0.24788356442042164	73	97
10	0.4216666666666667	1.1334248212076907	4.8205793161664525	0.8871169883362608	97	155
11	0.42	0.5177302928329837	4.134518484596067	0.6308299824018856	91	119
12	0.4116666666666667	0.9810123234314165	4.2869925	0.8952046472757862	61	51
13	0.43	1.1510712268232666	4.289856202403852	0.8780871391114339	48	51
14	0.4025	1.0017145184597183	4.157908265277858	0.71753921	58	70
15	0.43	1.3569982206657905	3.9694392049814273	0.72386471	54	50
16	0.4241666666666667	0.9189939201669776	1.9662197319213583	0.7682504306417545	63	160
17	0.42583333333333334	1.4922800976465649	4.390186488708082	0.6123056011425922	41	71
18	0.42	1.5380165409082498	3.756623291939448	0.7563778025493537	58	106
19	0.4116666666666667	0.9218166189444703	1.0439567569750845	0.5683200850516896	80	149
20	0.455	2.176362856512216	4.398097492290283	0.8030163739383746	42	63
21	0.41	0.8868106763205024	1.1259309906544481	0.5615593391117423	73	140
22	0.4025	0.8617750354947239	1.3914593952270073	0.5474142208838879	61	140
23	0.4141666666666667	0.8188661259313526	1.0760362651887831	0.5348694051090503	76	140
24	0.4116666666666667	1.0479450458918316	1.5774402241861316	0.40961886401122527	66	170
25	0.4225	1.3727599183232493	1.3934259852090334	0.6894554999214544	56	141
26	0.42083333333333334	1.5789359110410712	1.9742720611829516	0.5724654226972935	73	170
27	0.405	0.7525733102572567	2.45153017	0.5184919418842308	49	135
28	0.43166666666666664	2.9905634610131133	2.8876515088066643	0.3683659691433314	44	113
29	0.42	0.7265019190414042	2.3880954152180056	0.5095370974981654	51	180
30	0.4266666666666667	2.140798524766157	2.3807706882976514	0.4693006475695917	48	131
31	0.4141666666666667	0.8387030975237171	1.391911126015841	0.6323496350739707	62	130
32	0.42083333333333334	0.5832358913040884	1.7366495065131762	0.5558494913971246	53	142
33	0.405	1.0276996467651855	1.2972332672193603	0.5939893436866489	58	135
34	0.4325	1.301144727244579	2.089526472527662	0.7058518129146886	58	110
35	0.4116666666666667	1.0890096610246829	2.6200538809793468	0.6573916448016859	34	130
36	0.4325	1.230944641770649	1.405668109115405	0.6048042743413055	38	151

37	0.42083333333333334	0.7055293832974876	2.1880279615187224	0.42037405622820123	47	87
38	0.41833333333333333	1.0069398560020288	1.7258544797378024	0.4836457500191549	20	165
39	0.4166666666666667	0.5083221917703109	2.9829162452626092	0.5137499985967875	67	124
40	0.43583333333333335	1.9264992662575098	3.400634437835968	0.4112456878231401	64	181
41	0.42	0.7880190635257001	1.2075931424605593	0.58904277	71	141
42	0.40166666666666667	0.8812899911986412	1.2107849139232842	0.8181636433351727	59	133
43	0.4216666666666667	0.6426140959734903	1.2863975750021484	0.8066950788093578	55	119
44	0.43166666666666664	1.1376858759148494	1.6376326004506536	0.8184547489400937	60	101
45	0.43	0.9916144129670761	3.2168650446196545	0.7408472696225467	27	134
46	0.42333333333333334	0.7652410013510159	1.5253494274926964	0.8356763557516627	51	118
47	0.43833333333333335	1.4298758582815696	3.616973433323522	0.6926021213570847	45	148
48	0.4091666666666667	1.2227223424705917	1.903777594318016	0.6618901444994358	58	159
49	0.4083333333333333	0.6396644341916279	2.2570354176103953	0.6361789718062987	65	126

The Bayesian method with Optuna for the Ant Colony Optimization (ACO) was conducted over 50 iterations, resulting in the following optimal hyperparameters configuration:

- **Number of Ants (num_ants):** 59
- **Number of Iterations (num_iterations):** 133
- **Alpha (α):** 0.8812899911986412
- **Beta (β):** 1.2107849139232842
- **Evaporation Rate:** 0.8181636433351727

The best fitness achieved with these hyperparameters was 0.40166666666666667.

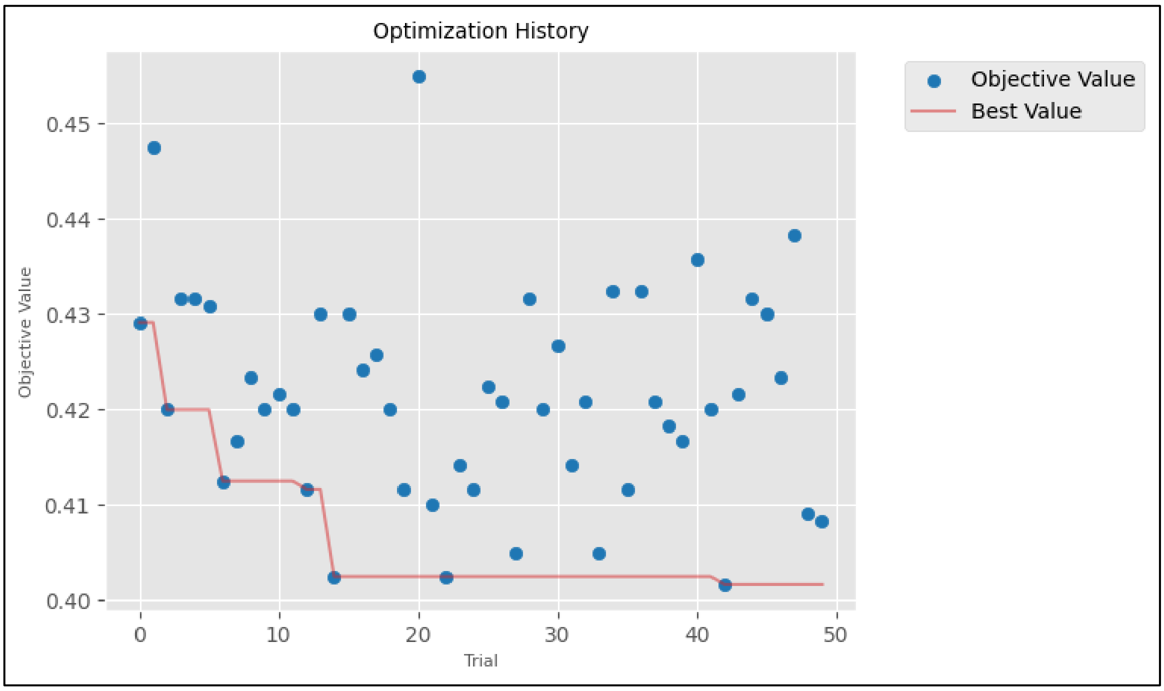

Figure 14. Optimization History for hyperparameters for Ant Colony Optimization.

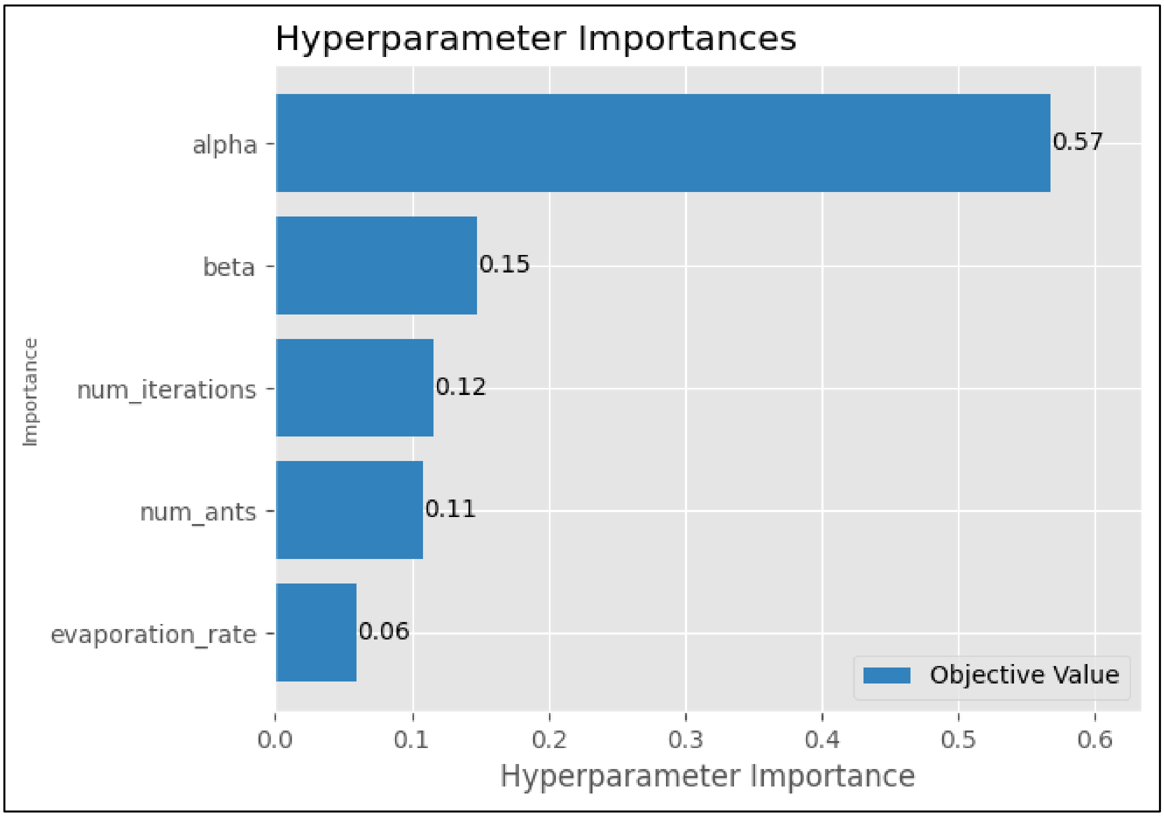

Figure 15. Hyperparameters importances of Ant Colony Optimization.

This figure displays the relative importance of the hyperparameters in Ant Colony Optimization. The alpha (α) parameter contributes 57% to the optimization process, while the beta (β) parameter accounts for 15%. The number of iterations is 12%, the number of ants is 11%, and the evaporation rate contributes 6% to the overall performance.

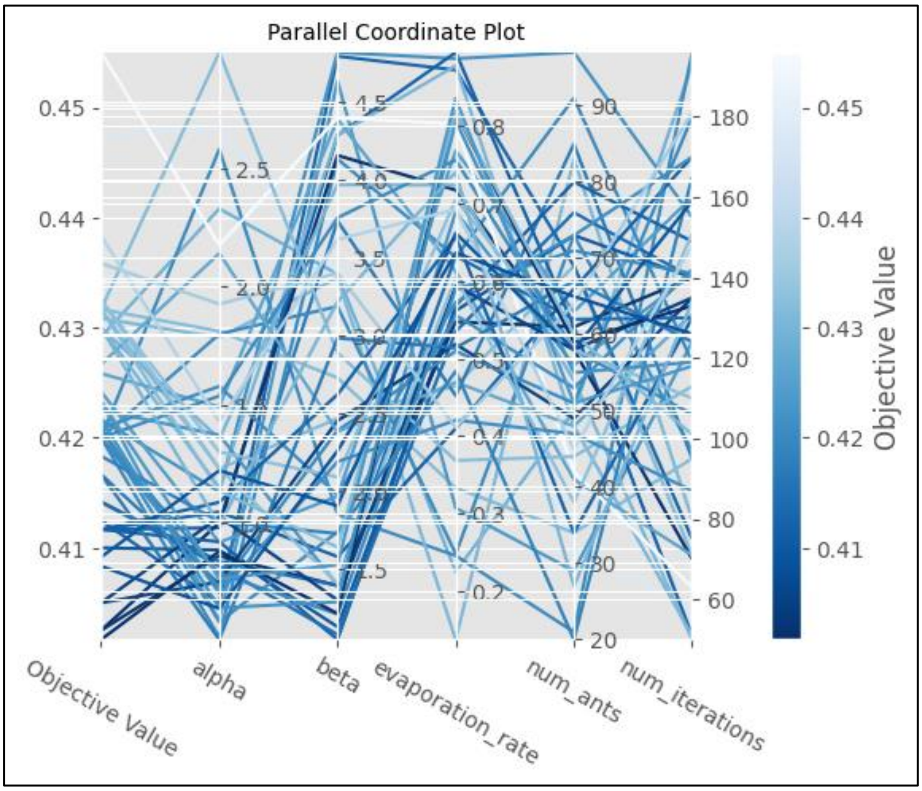

Figure 16. Parallel Coordinate plot for ACO.

This figure illustrates the parallel coordinate plot for Ant Colony Optimization (ACO). The plot visually represents the relationships and trade-offs between different hyperparameters used in the ACO algorithm. Each line in the plot corresponds to a specific trial run, showcasing how various hyperparameter values such as alpha (α), beta (β), number of iterations, number of ants, and evaporation rate contribute to the fitness value. This visualization helps in understanding the impact of each hyperparameter on the performance of the algorithm and in identifying optimal combinations for improved results.

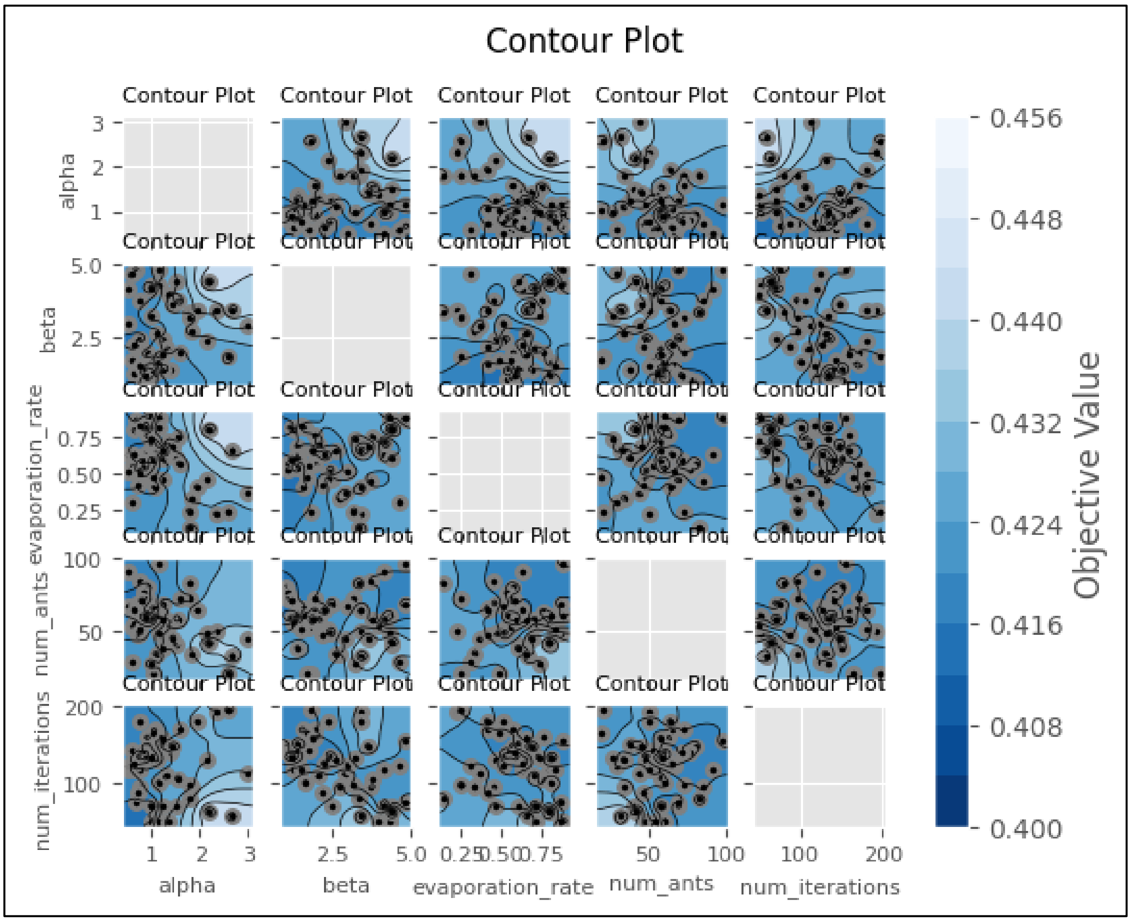

Figure 17. Contour Plot for ACO.

This figure depicts the contour plot for Ant Colony Optimization (ACO). The contour plot represents the relationship between two selected hyperparameters at a time, showing how different combinations affect the fitness value. Each contour line indicates regions of similar fitness values, enabling us to identify optimal ranges and interactions between the hyperparameters. This visualization is crucial for understanding how changes in hyperparameters such as alpha (α), beta (β), number of iterations, number of ants, and evaporation rate influence the performance of the ACO algorithm. The contour plot helps in fine-tuning these parameters to achieve better optimization results.

Table 2.1. Best Results for each iteration with different seeds for ACO.

seed	best_fitness	worst_fitness	average_fitness	median_fitness	std_dev_fitness
1	0.44416667	0.54833333	0.49077684	0.48916667	0.019348
2	0.43916667	0.54	0.49074859	0.48833333	0.02118
3	0.455	0.53583333	0.49187853	0.49166667	0.019029
4	0.42333333	0.53833333	0.47964689	0.48	0.020723
5	0.44	0.56833333	0.49004237	0.48833333	0.023548
6	0.4275	0.54083333	0.49067797	0.4925	0.021811
7	0.4475	0.56166667	0.49686441	0.49333333	0.02544
8	0.42833333	0.54916667	0.47822034	0.48	0.023466
9	0.46	0.56	0.50289548	0.5	0.019669
10	0.44083333	0.54	0.49388418	0.49166667	0.021671
11	0.455	0.55166667	0.50384181	0.50166667	0.022203
12	0.445	0.5475	0.4884322	0.4875	0.022992
13	0.43166667	0.53833333	0.49163842	0.495	0.024837
14	0.4375	0.555	0.49230226	0.49166667	0.024888
15	0.4225	0.5225	0.47718927	0.48083333	0.025022
16	0.42416667	0.51833333	0.48632768	0.48916667	0.021581
17	0.40833333	0.55583333	0.49361582	0.4925	0.027542
18	0.43333333	0.5425	0.48127119	0.48	0.023023
19	0.4475	0.5375	0.49252825	0.4925	0.020151
20	0.45916667	0.56333333	0.50139831	0.50083333	0.022838
21	0.4375	0.53916667	0.48950565	0.49	0.02165
22	0.45666667	0.5425	0.49679379	0.49833333	0.02116
23	0.44166667	0.54083333	0.49370056	0.49583333	0.021362
24	0.43666667	0.54916667	0.48679379	0.48916667	0.020206
25	0.46416667	0.5775	0.50850282	0.50916667	0.020111
26	0.42666667	0.54166667	0.49009887	0.49	0.023865
27	0.45833333	0.53416667	0.4909322	0.49083333	0.018706
28	0.4425	0.53833333	0.48415254	0.48333333	0.020361
29	0.45666667	0.55	0.49313559	0.49333333	0.019734
30	0.46	0.52916667	0.49370056	0.49083333	0.019025
31	0.415	0.52333333	0.47800847	0.47833333	0.023113

This table shows that the Best Results were gotten with seed 17 for all 31 iterations for the ACO. Besides that, you can see final results for all seeds are of near values between each other which indicates the best hyperparameters configuration seems appropriate with the results. Results are not as good as with GA, but that is to be expected since ACO relies on pheromone trails to guide the search process. While effective for exploitation (intensification of search in promising areas), it can sometimes lead to premature convergence if not enough exploration is performed. The pheromone update mechanism can overly favor certain paths, reducing diversity in solutions. While ACO can also be parallelized, the need for frequent pheromone updates and the interdependence of ants' paths can make it less straightforward to implement efficiently for very large problems.

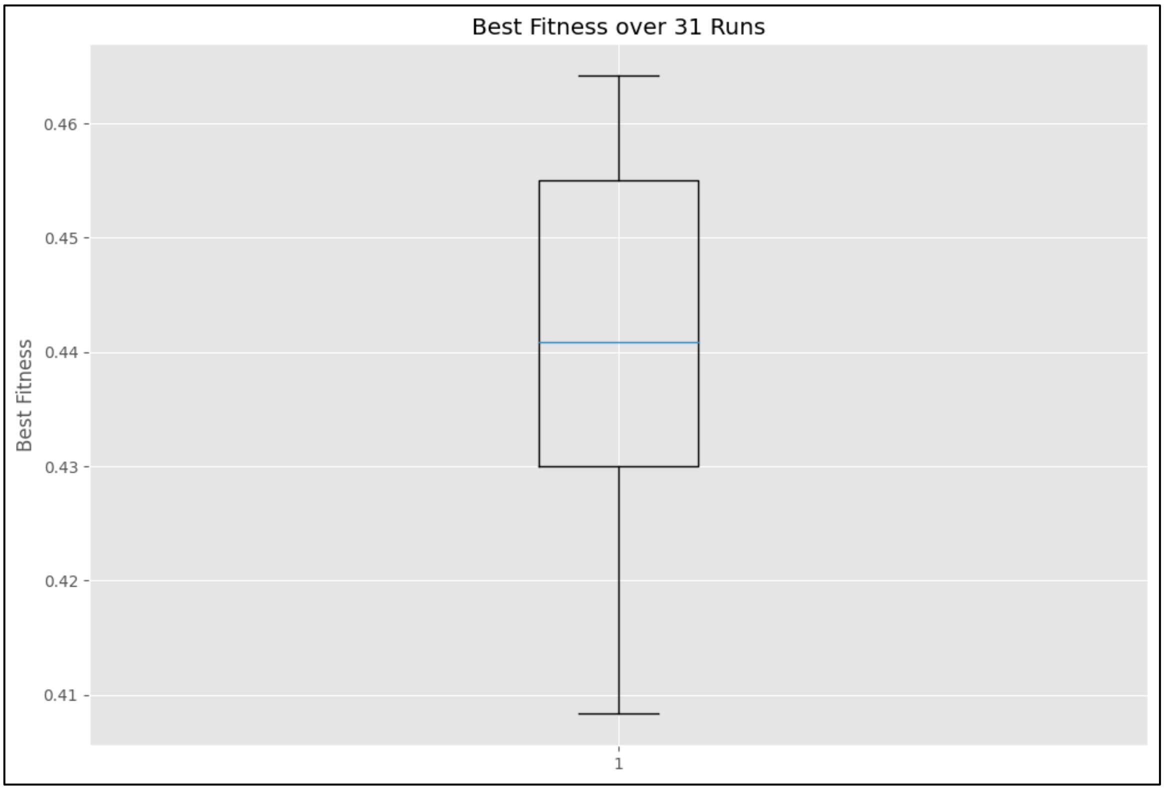

Figure 18. Best Fitness Over 31 Runs.

This figure illustrates the best fitness values obtained over 31 different runs of the Ant Colony Optimization (ACO) algorithm. Each run was initialized with a different seed to ensure variability and robustness in the results. The Wilcoxon signed-rank test p-value is 9.313225746154785e-10, indicating a statistically significant difference in the best fitness values across the runs. This test is used to compare the differences between two related samples and assess whether their population mean ranks differ.

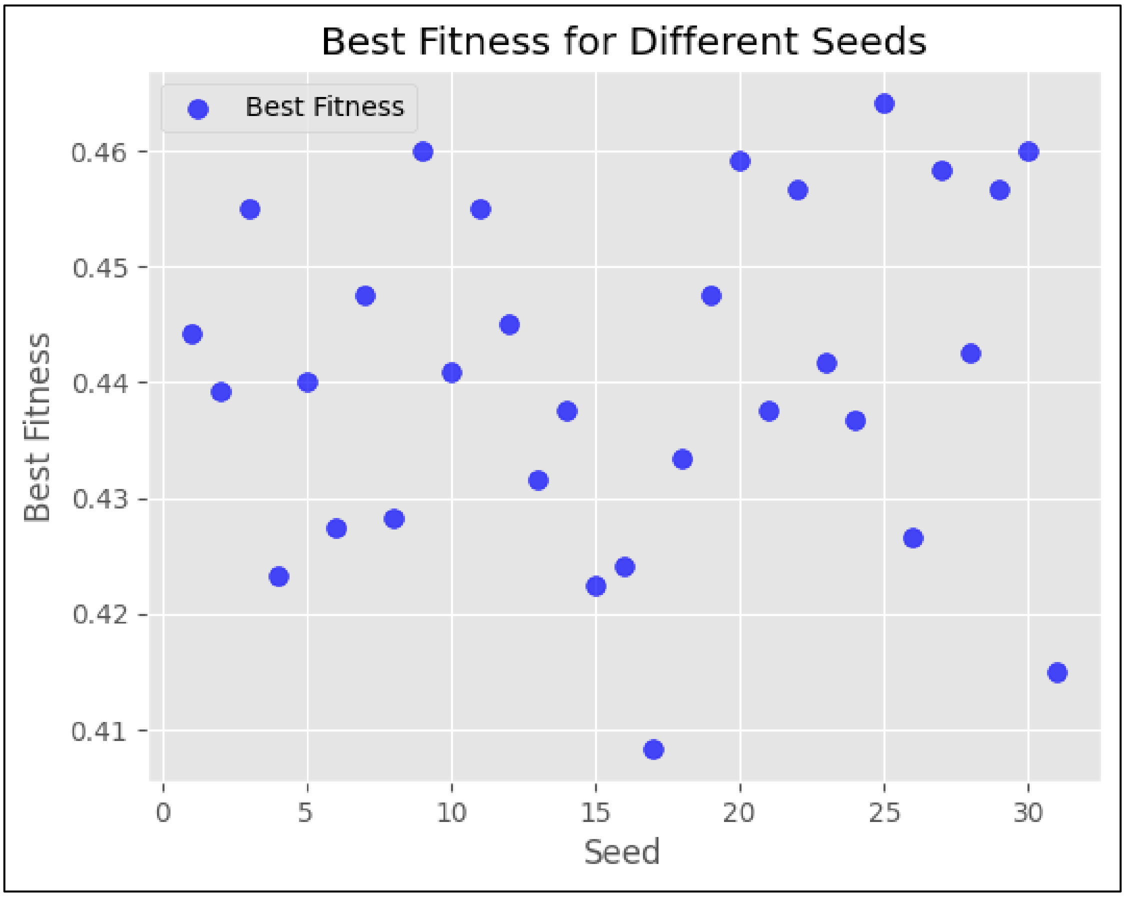

Figure 19. Best Fitness for all 31 different seeds for the ACO.

This figure displays the best fitness values achieved for each of the 31 different seeds used in the ACO runs. It shows the variation in performance due to the stochastic nature of the algorithm and the impact of different initializations on the final results.

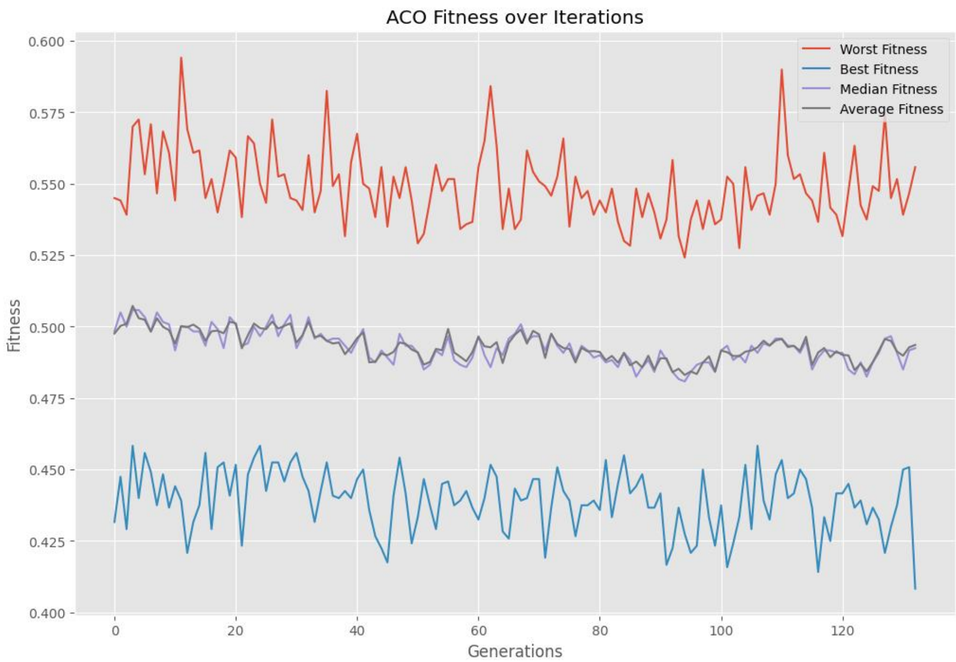

Figure 20. ACO fitness over population.

This figure plots the fitness values of the ACO algorithm over different population iterations. It provides insights into how the fitness of the solutions evolves as the algorithm progresses, showcasing the convergence behavior and the improvement in solution quality over time.

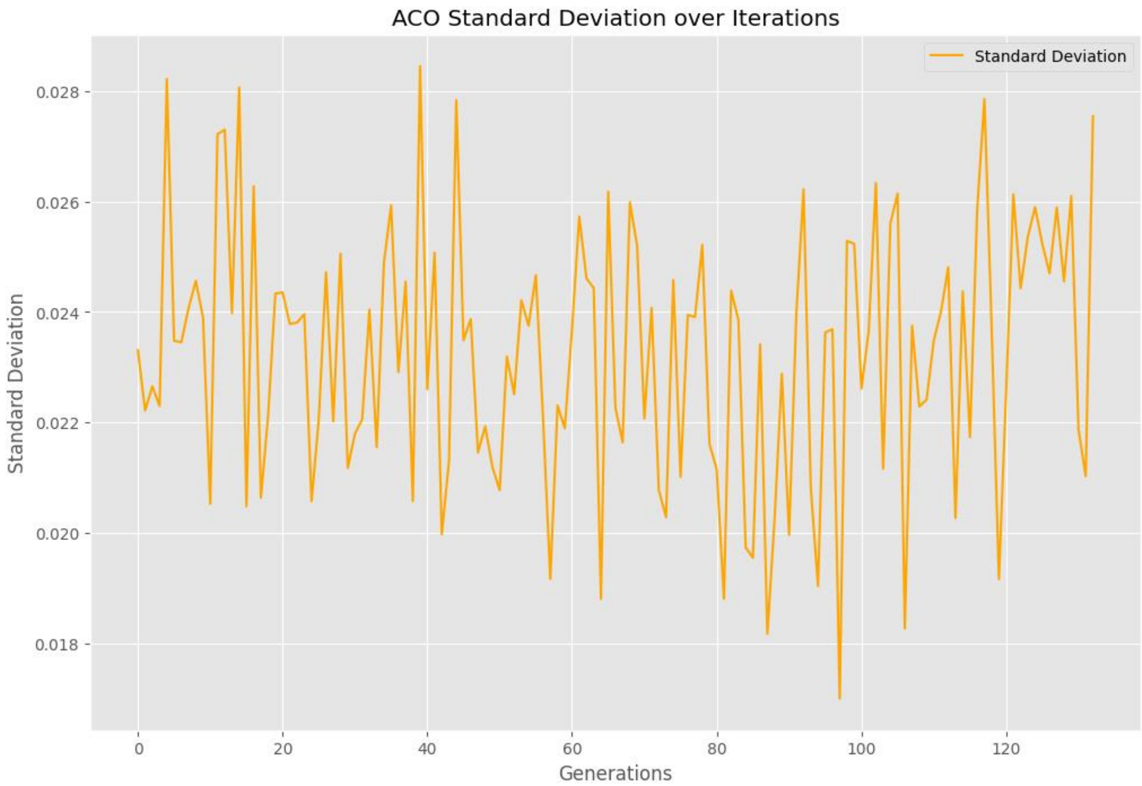

Figure 21. ACO Standard Deviation over Iterations.

This figure illustrates the standard deviation of the fitness values over the iterations of the ACO algorithm. It indicates the variability in the fitness values as the algorithm progresses, highlighting the stability and consistency of the solutions generated at different stages.

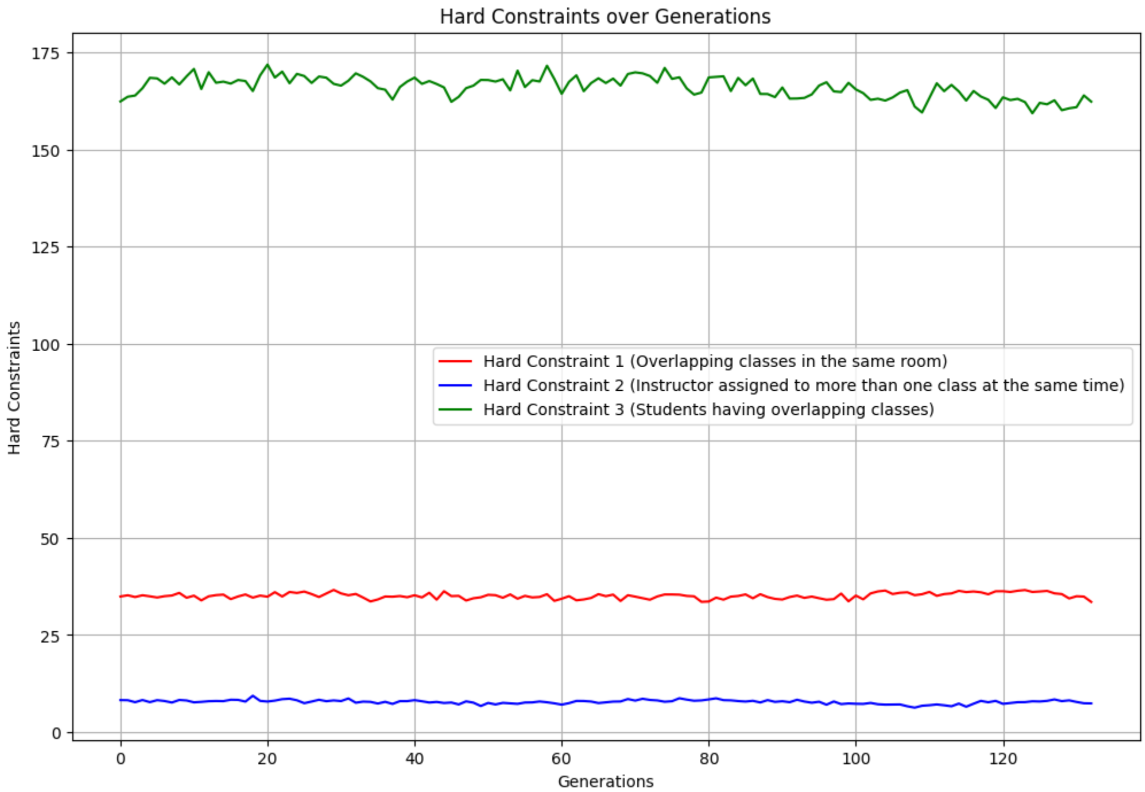

Figure 22. Hard Constraints over generations for the ACO.

This figure shows the compliance with hard constraints across different generations of the ACO algorithm. It tracks how well the solutions meet the predefined hard constraints, such as avoiding overlapping classes in the same room, preventing instructors from being assigned to more than one class at the same time, and ensuring students do not have overlapping classes.

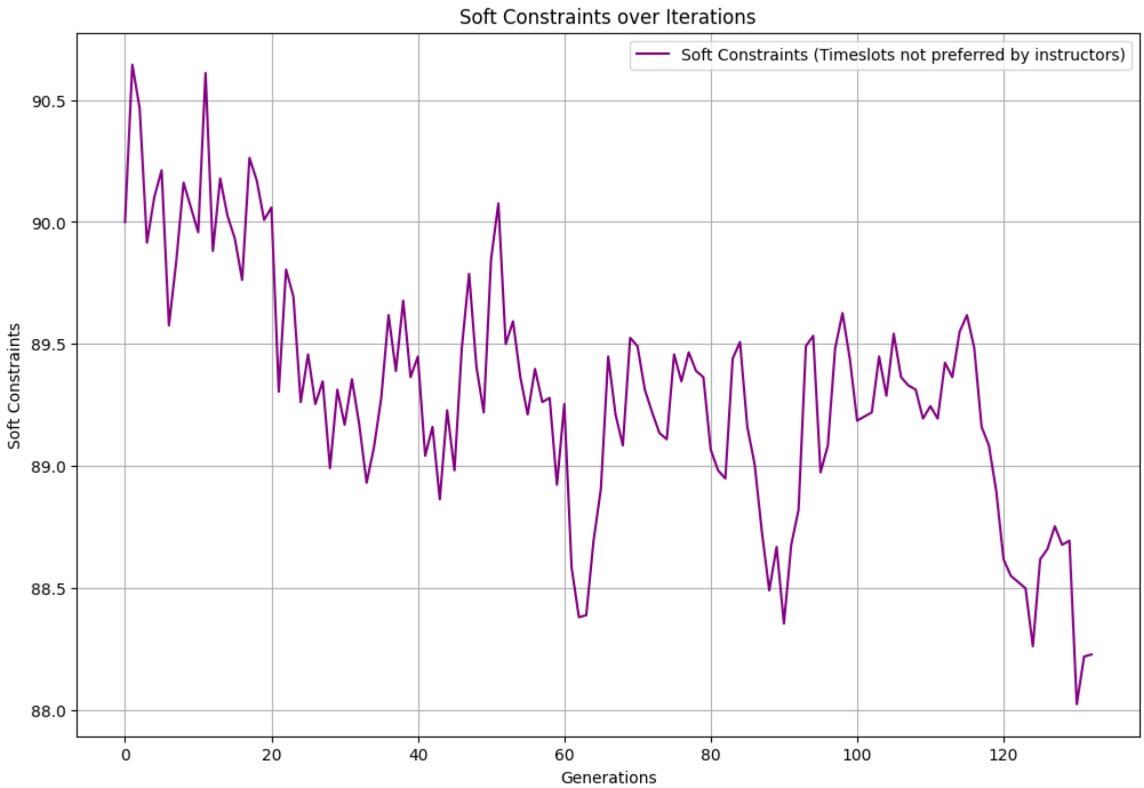

Figure 23. Soft Constraints over generations for the ACO.

This figure presents the compliance with soft constraints over the generations. It specifically focuses on the instructor preference compliance, which is considered a soft constraint. The figure tracks the percentage of times the solutions meet the preferred timeslots of instructors.

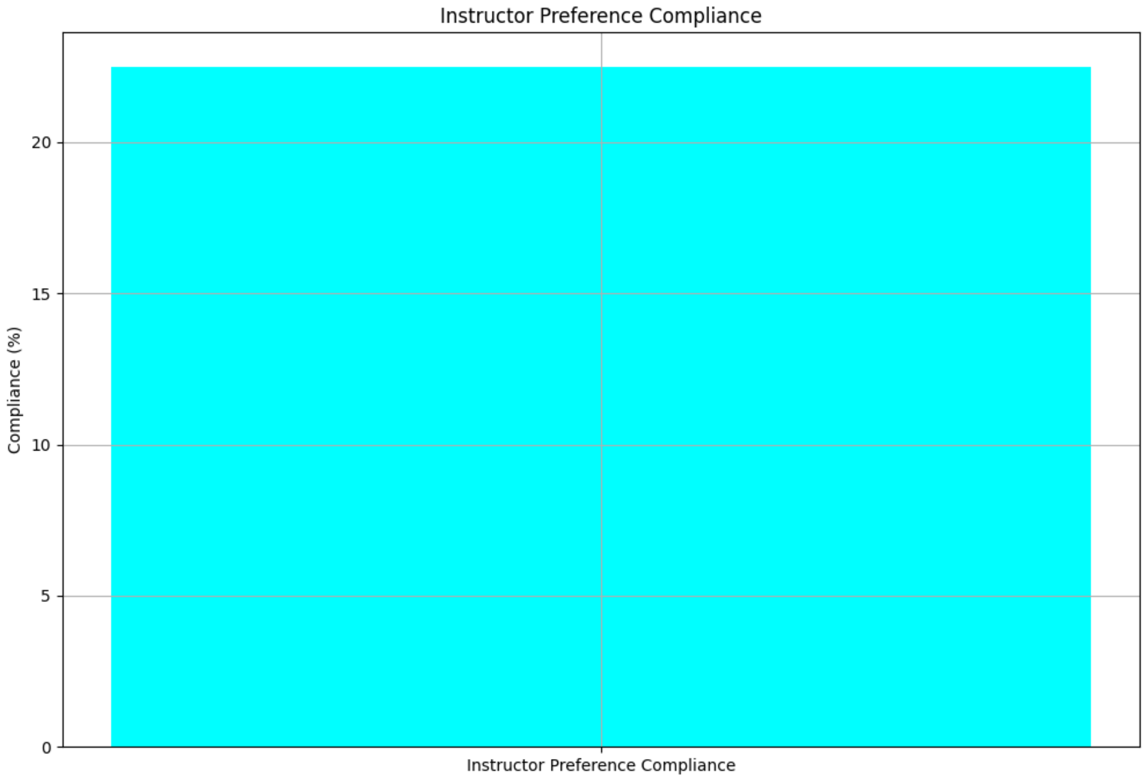

Figure 24. Soft Constraint Compliance. Instructor Preference Compliance: 25.00% (Soft Constraint 1)

This figure provides a detailed view of the compliance with the soft constraint related to instructor preferences. It indicates that 25.00% of the time, the generated schedules meet the preferred timeslots of the instructors.

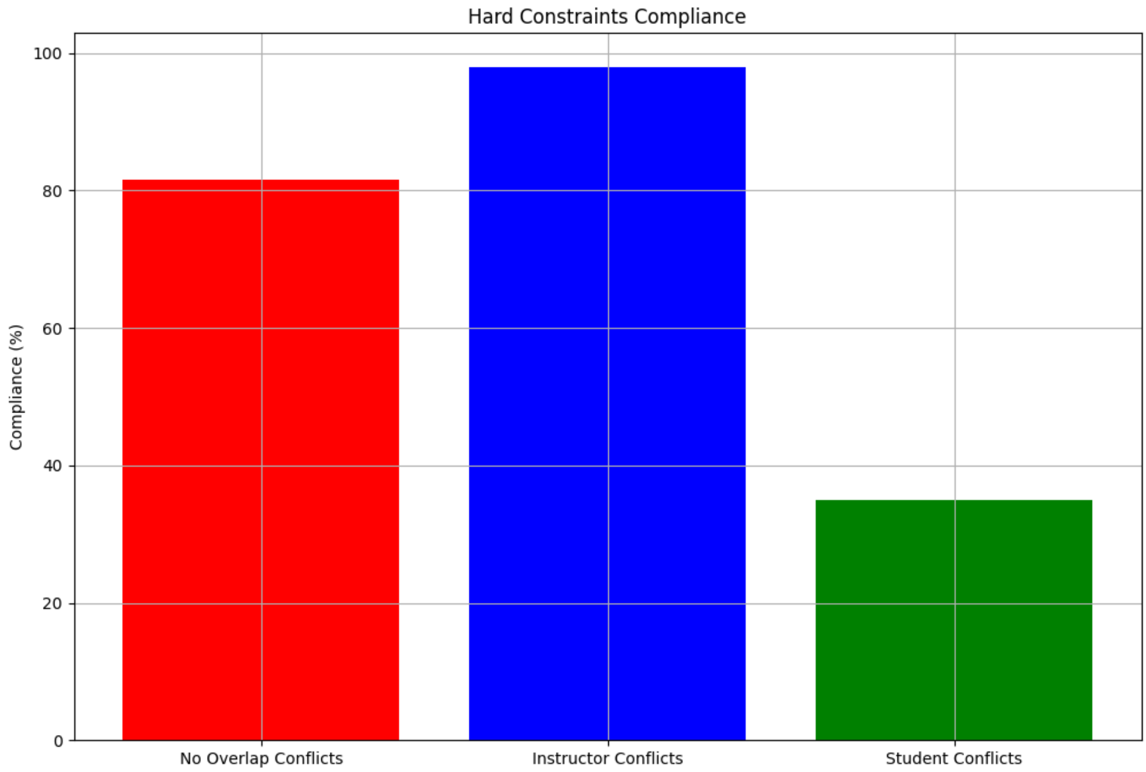

Figure 25. Hard Constraints Compliance for the ACO.

This figure summarizes the compliance with all hard constraints for the ACO algorithm. The results show:

- Compliance with no overlap conflicts (Hard Constraint 1): 82.50%
- Compliance with instructor conflicts (Hard Constraint 2): 98.00%
- Compliance with student conflicts (Hard Constraint 3): 41.50%

It is to note that all these graphs were plotted with the results for the ACO for the best hyperparameters and best performing seed for ACO (17)

Overall Results for ACO:

- **Instructor Preference Compliance:** 25.00% (Soft Constraint 1)
- **Compliance with no overlap conflicts:** 82.50% (Hard Constraint 1)
- **Compliance with instructor conflicts:** 98.00% (Hard Constraint 2)
- **Compliance with student conflicts:** 41.50% (Hard Constraint 3)
- **Best Fitness or Path for the ACO:** 0.4083333333333333

These results indicate that the ACO algorithm performed well in terms of hard constraints, especially in avoiding instructor conflicts. However, there is room for improvement in student conflict resolution and instructor preference compliance. The best fitness value achieved demonstrates the efficiency of the ACO algorithm in finding optimal or near-optimal solutions for the course timetabling problem.

Comparison of Genetic Algorithm (GA) and Ant Colony Optimization (ACO) Results

Results Summary

Genetic Algorithm (GA) Results:

- **Best Parameters:**
  - Population Size: 196
  - Maximum Generations: 122
  - Mutation Rate: 0.025167150341825872
  - Crossover Rate: 0.9847131373438699
  - Tournament Selection Size: 5
  - Best Trial: 13
  - Best Fitness: 0.1733333333333
- **Constraints Compliance:**
  - No Overlap Conflicts: 97.50%
  - Instructor Conflicts: 100.00%
  - Student Conflicts: 92.50%
  - Instructor Preference Compliance: 45.00%

Ant Colony Optimization (ACO) Results:

- **Best Parameters:**
  - Number of Ants: 59
  - Number of Iterations: 133
  - Alpha (α): 0.8812899911986412
  - Beta (β): 1.2107849139232842
  - Evaporation Rate: 0.8181636433351727
  - Best Fitness: 0.4083333333333333
- **Constraints Compliance:**
  - No Overlap Conflicts: 82.50%
  - Instructor Conflicts: 98.00%
  - Student Conflicts: 41.50%
  - Instructor Preference Compliance: 25.00%

Comparison Table

Metric	GA	ACO
Population Size	196	59 (Ants)
Maximum Generations/Iterations	122	133
Mutation Rate	0.025167150341825872	-
Crossover Rate	0.9847131373438699	-
Alpha (α)	-	0.8812899911986412
Beta (β)	-	1.2107849139232842
Evaporation Rate	-	0.8181636433351727
Best Fitness	0.1733333333333	0.4083333333333333
No Overlap Conflicts Compliance	97.50%	82.50%
Instructor Conflicts Compliance	100.00%	98.00%
Student Conflicts Compliance	92.50%	41.50%
Instructor Preference Compliance	45.00%	25.00%

Key Findings and Implications

- **Performance**: GA significantly outperformed ACO in terms of fitness value (0.173 vs. 0.408) and constraints compliance. The GA demonstrated higher effectiveness in generating feasible schedules that adhere to the defined constraints.

- **Constraints Compliance**: GA showed better compliance with all constraints, particularly with student conflicts (92.50% vs. 41.50%). This indicates that GA's exploration capabilities better handle the complexity of the problem.
- **Instructor Preference**: Both algorithms struggled with instructor preference compliance, but GA performed almost twice as well as ACO (45.00% vs. 25.00%).

Comparison of Hyperparameters Importance and Analysis

Hyperparameter	Genetic Algorithm (GA)	Ant Colony Optimization (ACO)
Crossover Rate / Alpha (α)	70%	57%
Population Size / Beta (β)	21%	15%
Mutation Rate	6%	-
Maximum Generations	4%	-
Number of Iterations	-	12%
Number of Ants	-	11%
Evaporation Rate	-	6%

Analysis

Genetic Algorithm (GA) Hyperparameters:

1. **Crossover Rate (70%)**: This is the most crucial parameter in GA, reflecting the importance of combining different solutions to explore new possibilities and maintain diversity in the population.
2. **Population Size (21%)**: A larger population allows more genetic diversity and a better chance of finding optimal solutions, but it increases computational cost.
3. **Mutation Rate (6%)**: Introduces new genetic material to the population, preventing premature convergence by maintaining diversity.
4. **Maximum Generations (4%)**: Defines the number of iterations the algorithm runs. This has a lower impact because GA relies heavily on the crossover and mutation operations within the population over generations.

Ant Colony Optimization (ACO) Hyperparameters:

1. **Alpha (α) (57%)**: Determines the influence of the pheromone trail on the selection of paths. High importance indicates that pheromone intensity significantly guides the search process.
2. **Beta (β) (15%)**: Defines the influence of heuristic information (like distance) on path selection. A balanced α and β ensure a mix of learned experience (pheromone) and problem-specific knowledge (heuristics).
3. **Number of Iterations (12%)**: More iterations allow ants to explore more paths and lay down pheromones, enhancing solution quality.
4. **Number of Ants (11%)**: More ants increase the exploration of the solution space, but too many can lead to higher computational costs.
5. **Evaporation Rate (6%)**: Controls the rate at which the pheromone trail evaporates, preventing convergence to suboptimal solutions by allowing exploration of new paths.

Challenges in Hyperparameter Optimization

One of the main challenges in optimizing hyperparameters was finding an efficient method. Initially, we tried Grid Search, but it proved too time-consuming and computationally intensive given our limited resources. Random search did not yield reliable results either. Manual tuning of hyperparameters was also attempted, but this approach lacked systematic rigor and efficiency. Eventually, we opted for Bayesian Optimization using the Optuna library, which provided a more structured and efficient search process.

At first, it was taking too much time to optimize the Genetic Algorithm Hyperparameters, so we decided to keep the tournament selection value fix at 5.

When optimizing the hyperparameters of the Genetic Algorithm (GA), the process can be very time-consuming due to the large search space and the need to evaluate multiple configurations. One of the hyperparameters, the tournament selection size, can significantly impact the performance of the GA. However, optimizing it alongside other hyperparameters would require considerable computational resources and time.

To mitigate this, the decision was made to fix the tournament selection size at a value of 5. This approach helps reduce the complexity of the optimization process by decreasing the number of parameters that need to be tuned simultaneously. A tournament size of 5 is a reasonable choice as it balances exploration and exploitation, providing a good trade-off for selection pressure.

By fixing this parameter, the focus could be placed on optimizing the remaining hyperparameters: population size, maximum generations, mutation rate, and crossover rate. This strategy not only speeds up the optimization process but also ensures that sufficient computational resources are allocated to fine-tune the most impactful parameters, ultimately improving the GA's performance efficiently.

Optimizing the hyperparameters for the Ant Colony Optimization (ACO) algorithm was particularly challenging. The process took over 10 hours on our local environment, and Google Colab often crashed before completion. This highlighted the need for substantial computational power and robust environments for hyperparameter tuning in complex optimization problems.

Additionally, we initially tried to implement the Genetic Algorithm (GA) without using the Inspyred library, relying solely on Pandas and NumPy. However, this approach resulted in overly complex and unwieldy code. We ultimately decided to revert to using the Inspyred library, which offered more manageable and efficient implementations of evolutionary algorithms.

Discussion

Analysis and Discussion

Performance: GA outperformed ACO in all hard constraint compliance metrics and in overall fitness value. GA's higher instructor preference compliance suggests better adaptability to soft constraints.

Hyperparameters Importance:

- **GA:** Crossover Rate (70%) was most crucial, followed by Population Size (21%).
- **ACO:** Alpha (57%) and Beta (15%) were the most significant, emphasizing the importance of pheromone influence and heuristic information.

Key Findings and Implications

The GA demonstrated superior performance in generating feasible schedules with high compliance to constraints. ACO, while effective, struggled with maintaining constraint compliance, particularly with student conflicts. This indicates that GA is more suitable for problems requiring strict adherence to multiple constraints.

Key Findings and Implications

- **Performance**: GA significantly outperformed ACO in terms of fitness value (0.173 vs. 0.408) and constraints compliance. The GA demonstrated higher effectiveness in generating feasible schedules that adhere to the defined constraints.
- **Constraints Compliance**: GA showed better compliance with all constraints, particularly with student conflicts (92.50% vs. 41.50%). This indicates that GA's exploration capabilities better handle the complexity of the problem.
- **Instructor Preference**: Both algorithms struggled with instructor preference compliance, but GA performed almost twice as well as ACO (45.00% vs. 25.00%)

Strengths and Weaknesses

GA Strengths:

- High exploration capabilities due to crossover and mutation operations.
- Robust performance across a wide range of parameter settings.
- Better scalability for larger problem instances due to population-based approach.

GA Weaknesses:

- Requires careful tuning of mutation and crossover rates to balance exploration and exploitation.

ACO Strengths:

- Effective at intensification of search in promising areas through pheromone updates.
- Can quickly find good solutions in early iterations due to heuristic information.

ACO Weaknesses:

- Prone to premature convergence if not enough exploration is performed.
- It takes a lot of time; it requires larger computational effort.
- Highly sensitive to parameter settings like evaporation rate and influence of pheromone vs. heuristic information.
- Less robust in maintaining diversity of solutions compared to GA.

The experiments highlight the importance of hyperparameter optimization in improving the performance of metaheuristics for complex problems like course scheduling. Bayesian Optimization with Optuna proved effective for this purpose. The results demonstrate the Genetic Algorithm's capability to generate feasible schedules with high compliance to hard constraints, although further work is needed to optimize for soft constraints. The visual analysis provides valuable insights into the algorithm's behavior and areas for improvement.

Conclusion and Future Work

Key Findings:

- GA outperformed ACO in terms of constraint compliance and overall fitness value.
- GA showed robustness across different seeds, indicating reliable performance.
- ACO struggled with student conflicts and instructor preferences, highlighting areas for improvement.

Implications:

- The effectiveness of GA in generating feasible schedules suggests its suitability for complex scheduling problems.
- ACO's performance indicates the need for better parameter tuning and constraint handling mechanisms.

Future Research:

- Hybrid approaches combining GA and ACO for initial exploration and solution refinement.
- Adaptive parameter tuning mechanisms to enhance algorithm performance.
- Advanced constraint handling techniques to improve compliance with soft constraints.
- Scalability improvements through parallel and distributed computing.
- Efficient resource utilization through parallel processing and cloud computing.
- Exploring other evolutionary algorithms like PSO for better performance.

References

- **GitHub:** https://github.com/IsmailCh0901/BAO-Project-Extraordinaria-/blob/main/README.md

- **Google Collab :** https://colab.research.google.com/drive/102iMOTvGnsjAuWj2xyNnBeRWupHt0LXd?usp=sharing#scrollTo=p7o7BQD8mT_S

List all the academic papers, books, and other resources cited in the document. Follow a consistent citation style throughout.

1. Repository UAEH. (n.d.). Investigación de métodos de optimización en la programación de horarios. Retrieved from https://repository.uaeh.edu.mx/revistas/index.php/investigium/article/download/9866/9736/
2. UTM. (n.d.). Tesis digital. Retrieved from http://jupiter.utm.mx/~tesis_dig/6557